Code and Poetry An Exploration of Logic throughout Art, Computation and Philosophy

Rosi Grillmair
Kunstuniversität Linz

Kunstuniversität Linz
Universität für künstlerische und industrielle Gestaltung
Interface Cultures
(Institut für Medien)

Code and Poetry

An Exploration of Logic throughout Art, Computation and Philosophy

von Roswitha Magdalena Grillmair

zur Erlangung des akademischen Grades
Master of Arts

Betreut von:
Univ. Prof. Dr. Laurent Mignonneau
Univ.-Ass. Fabricio Lamoncha Martinez, MA
PhD Maša Jazbec

Linz, September 2019

I wanted to create something real,
but I restricted myself to simulations.
Therefore, I wrote my thesis about simulations,
and it became something real.

I personally thank my supervisors Laurent Mignonneau, Fabricio Lamoncha and Maša Jazbec, as well as Christa Sommerer and Manuela Naveau.

My parents, Amir Bastan, Gabriella Gordillo, Maja Burja and Katharina Edelmair and everyone at the department of Interface Cultures - through you I could grow personally and academically.

On the shoulders of many giants, I want to explicitly mention those, whose works were crucial for this thesis: Josha Bach, Annette Vee, Gottlob Frege, Cristina Videira Lopes, Sandra Trostel and Douglas R. Hofstadter.

Table of Contents

Abstract **7**

1. Field of Research **8**

1.1. Code 8

1.2. Poetry 9

1.3. A Comparison of Programming and Creative Writing 10

1.4. Code Poetry 14

1.5. Language 16

1.6. Computation and Simulation 18

2. A History of Logic in Mathematics and Philosophy **21**

2.1. Propositional Logic 22

2.2. The Foundations of Mathematics 26

2.3. Formals Systems 27

2.4. “Begriffsschrift” by Gottlob Frege 27

2.5. “Principia Mathematica” by Bertrand Russell and Albert N. Whitehead 28

2.6. The Incompleteness Theorem by Kurt Gödel 28

2.7. Computational Machines 29

2.8. Ludwig Wittgenstein: A Deterministic Language 30

2.9. Boolean Logic 31

2.10. Programming Languages 31

2.11. The Artificial Mind 31

3. Artistic and Scientific Methods **33**

3.1. Infinity 33

3.2. Self-Reference 34

3.3. Recursion 34

3.4. Repetition 35

3.5. Paradox 35

3.6. Nothingness 36

4. Artistic References **37**

4.1. “Variable” by Maja Burja and Urška Aplinc 37

4.2. “Hybris” by Arjan Brentjes 39

4.3. The Useless Machine - A Poetic Object 41

4.4. Design Studio Mischer’Traxler 42

4.5. Exercises in Programming Style 44

4.6. Techniques of Code Poetry 45

4.6.1. Subversion 45

4.6.2. Esoteric Programming Languages 46

4.6.3. Quine 48

5. Personal Project: FURTHER, FURTHER, FURTHER **50**
5.1. Project Outline 50
5.2. Participants 51
5.3. Content 52
5.4. Theoretical Background 55
5.4.1. The Text “A Repetitive Process” (see Appendix 3) 56
5.5. Development 57
5.6. Documentation 59
5.7. Analysis 69
5.7.1. Mimicry 69
5.7.2. Repetitions 71
5.7.3. Self-Reference and the Effect of Repetition 72
5.7.4. Information 73
5.7.5. Narrative by Repetition 74
5.7.6. Collective Construction of Art 75
5.7.7. Space exploration: How to locate information in space 76
5.7.8. Learning 77

6. Personal Project: For while { Exception println #update let !doctype @type } ; **79**
6.1. Project Outline 80
6.2. Participants 81
6.3. Ways of Analysing a the Code Poems 82

7. Conclusion **85**

8. Bibliography **87**

9. Image References **89**

10. Appendix **90**
10.1. Texts about “Variable” by Maja Burja and Urška Aplinc 90
10.2. Interview with Maja Burja and Urška Aplinc about their work “Variable” 91
10.3. Text “A Repetitive Process” by the Author 93
10.4. Anonymous Code Poems 94
10.5. Visual Code Poems 98

Abstract

This is an invitation to all readers to join me in my exploration of logic and art. The preparation for it is presented throughout the next chapters:

- First, the space for the exploration is located:
  Code and Poetry, Language, Computation and Simulation
- Now readers are invited to equip themselves with the necessary tools of the exploration: **A Tool Set of Logic.**
- Then follows an introduction to the creative use of these tools:
  Paradoxes, Repetition, Construction and Destruction of Rule Systems
- Readers are getting ready for the exploration and are presented other artists who use similar tools and/or work with the same materials.
- The exploration begins and ends here.

This thesis aims to put a finger on the overlay between mathematics and philosophy. Its method is to show that art and logic come together through language.
To put the reading of this thesis into the right context, some things need to be stated at the beginning. Firstly, when “code” is mentioned, it stands for mathematics, for computation in specific. The second premise is that “poetry” stands for the creative use of language and propositions.
My interest from an artistic and analytical point of view is in the subversion of the two: If logic is the principal mode for exploring math, philosophy, computation and language - how can it be played against itself in order to spark artistic, original thought?

The exploration will be accompanied by a constant reality-check:

- How much decoration is needed to talk about *art*?
- How many rhymes to recognize *poetry*?
- How many lines of code are needed to describe *computational thinking*?
- How many philosophers are quoted in order to understand what *logic* is?
- Which equations should be formed to make clear what *math* is?
- And how many translations are needed to get the concept of *language*?

These six lines of thought revolve around one center point. The underlying concept of them all becomes more apparent when two or more lines intersect: A code poem, a logical fallacy in language, a paradox noted in symbols and in prose, the concept of repetition acted out on stage.

It’s an ever-lasting examination of logic and a game of thought, a discovery process of the mind that applies reason and finds art. I wish all readers a pleasant journey!

1. Field of Research

My field of research is the interplay of code, poetry, language, computation and simulation. Code and computation thereby form the functional basis, the logical system that any further explorations need.
Poetry stands for the artistic act, the art work that can be consumed in a certain time frame, based on instructions that were set before. Language - sometimes expressed in the written word - is the medium which arises throughout code and in poetry. Computation is needed to prove experiments: Through computed simulations, it is possible to observe how code and poetry work together through language.

1.1. Code

A code is a rule system comprised of signals and symbols to communicate information[1] - in the context of this book, the words coding, scripting and programming can be used interchangeably.

In the broadest sense, the word code and language mean the same thing. Being able to participate in the exchange of information through a code means understanding the rule system and applying it. Thereby information can be sent (encoded) and received (decoded).

Code plays an important role in the development and expression of natural language as well as social behaviour and in the identification within a group through the use of encoding and decoding messages.[2] Nevertheless, in this thesis, another definition of code will be used - that puts the social aspect aside.

The code referred to in this thesis is defined as a formal language, unlike natural languages it does not focus on communication but on its mathematical use. This form of code is extensively used since the 20th century for automated computation.[3]

Before that, formal languages existed as models for correct reasoning: the logic applied to mathematics as well as philosophy were expressed via modular propositions: each proof was based on something that was proven before.
If the rule of inference by correct reasoning are exercised carefully, (“the code is executed correctly”) a formal language becomes consistent and deterministic. (Cryan, 2016)

[1] code. 2019. In Merriam-Webster.com. Retrieved August 18, 2019, from https://www.merriam-webster.com/dictionary/code
[2] Krauss, R., & Chiu, C.-Y. (1997). Language and social behavior. In D. Gilbert, S. Fiske & G. Lindsey. Handbook of Social Psychology, 4(2), 41–88. https://doi.org/10.1177/003803857300700340
[3] Church, A., & Turing, A. M. (1937). On Computable Numbers, with an Application to the Entscheidungsproblem. The Journal of Symbolic Logic, 2(1), 42. https://doi.org/10.2307/2268810

The meaning of programming in the context of this text can be summarized in three points.

1. Programming means to write code that instructs a computer. It takes the conditions of the computer into account. How much memory is allocated for the program? How long does the machine need to process?
2. Programming also means a form of philosophical exploration. Formal systems are built by the symbols of language in order to reason about our world.
3. Programming is a form of writing with a certain syntax, style, sentence structure and audience that the writer tires to appeal to.

1.2. Poetry

According to the dictionary, poetry is considered a form of literary art in which language is used for its aesthetic and evocative qualities. It contains multiple interpretations and therefore resonates differently with each reader.[4]
For this text it should be defined in a broader sense, outside of literature and as a genre of art.

A written poem.
Symbols of written language are the medium that this kind of poetry is based on. The poet knows the language, its structure, its rhythm, its grammar and applies them in order to create the poem. The poem is then perceived through reading: It is structured by letters, words and paragraphs in a predefined sequence. The reader consumes the poem by following the preset steps.

A Performance or Dance Choreography
This kind of poetry is based on a sequence of body movements. Its symbols are gestures and mimic. The poet or choreographer knows body language, expressive body movement as well as rhythm and creates predefined sequences of them that can be acted out. The viewer consumes the performance again in a preset order of events.
The performer is following the score and thereby translates the rules that were set by the choreographer.

Example: Fluxus Event Score
The art group Fluxus wrote event scores that used the notation of everyday language that were meant to be executed and thereby performed by anybody who was able to read the score. “Proposition” by Alison Knowles (1962) is a set of these instructions. One of them called “Make a Salad” instructs the performer to do exactly that as an artistic act. [5]

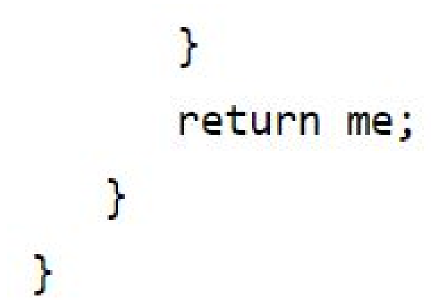
[4] poetry. 2019. In Merriam-Webster.com. Retrieved August 18, 2019, from https://www.merriam-webster.com/dictionary/poetry
[5] Anderson, V. (2016). Fluxus : Event Scores and Their Performance.

A poetic object
All steps in the production of an object are known and were studied by the artist. The rules of production are set by the artist, and they can be observed by looking at the finished object. The way it was produced follows certain rules, a manual for example. When the viewer is presented with the object, they can see a record and deduce how it was produced by observing it. The object accurately represents the reason for its own existence.

Example: Surrealist Object - Assemblage “A three-dimensional work of art made from combinations of materials including found objects or non-traditional art materials.” like “Object” by Meret Oppenheim (1936) [6]

Computer Generated Art
The artist knows the rules of programming and the relation of code and its visual/auditory output. For a piece of generative art, the instructions of how to execute it are formalized in code. When the computer executes the program, it does that in a strictly predefined sequence. The observer is then presented with that sequence as a visual or auditory result.

1.3. A Comparison of Programming and Creative Writing



```
          }
          return me;
      }
  }
```

Fig 1: Last lines of a code poem by Álvaro Matías Wong Díaz written in the programming language Java from the book “code {poems}” by Ishac Bertran (2013)

Both programming and poetry are forms of writing. This chapter summarizes arguments for programming as a form of creative writing.

“Programming is an aesthetic as well as functional activity.”
(Vee, 2017, 124)

According to Annette Vee, who authored “Coding Literacy - How Computer Programming is changing Writing” code is written on the intersection of human and machine interaction. It has two functions and two audiences: The descriptive function (Readability) serves the human reader while the performative function (Computability) serves the machine which is instructed to execute the code.

[6] Assemblage. 2019. In MoMA.com. Retrieved September 19, 2019, https://www.moma.org/learn/moma_learning/glossary/#assemblage

Programming should be “clear and consistent for both the computer and the human reader, omit unnecessary processes for the computer and make active processes transparent to the human reader.” (Vee, 2017, 124)

Additionally, form and meaning merge within programming. Its performative and computational functions deliver a unique possibility that poetry can not: it is written to represent a task and meant to execute the task at the same time. This “executability” of a task with a predefined result is missing from poetry.
The program construct, unlike the poet's words, is real in the sense that it moves and automates, producing observable outputs separate from the construct itself.

One can argue that poetry is executed on the reader's mind as well. The outcome of the execution on someone's mind is quite open and its form and extent left to the reader.

The Functional Language of Code and Poetry

If we consider both a piece of program code and a poem as forms of functional language, we will find that the code is much more direct in its functionality. Its existence is justified by necessity, even if it is not directed to a single person. Code is language without the need of communication. Without communication it still fulfills its performative function for the machine. If it does not fulfill a communicative or artistic purpose, it becomes part of the machine that is executing it. It stores information and triggers data flow.

A piece of code is entirely instruction based and reveals a predefined purpose. The sender is responsible for explaining the purpose of the written code. In a poem, the sender is only responsible for delivering the text, but its impact and purpose are defined by the receiver. The code requires a receiver who is ready to decode after a predefined rule set. The poem requires a reader who is ready to perceive undirected pieces of information and decode them with a personal rule system.

The Creative Potential of Code and Poetry

“Writing and Coding are creative acts, yet we try to label one as engineering (as if engineering weren’t creative).” (Vee, 2017, 123)

ON/OFF or zero and one eventually substituted exchange of cables in a computer. By now written code is substituting them. The machine could still read and execute instructions if it was a sequence of binary numbers or electronically as logical gates that process signals.[7]

So code is written for human consumption and the author makes decisions on the architecture and structure of the algorithm as well as its design.
In order to write more efficient algorithms and do so in collaborations, humans needed a more intuitive form of communication, closely related to spoken language. This language

[7] Vee, A. (2017). Coding Literacy: How Computer Programming is Changing Writing. In Software Studies.

should be read like sentence-based instructions for the computer in imperative form, so even non-coders might understand what is happening within the code.

Thus, the development of programming languages was a necessary step. The writer makes choices of how to structure their code and how to communicate its intent both to the computer and other human readers.
It thereby becomes a genre of writing. This can be easily proven by comparing how writers would all describe a situation differently. The same is true for coders: every coder writes their programs differently, even though they want the computer to execute the same task.

Ishac Bertran, a Barcelona based artist and coder is one of the pioneers who experimented with code as a creative form of writing and supported the niche genre “Code Poetry”.
A code poem is written like an algorithm: its style and functionality originate in source code. The code poem is executable, and thereby technically correct, and serves an artistic purpose. In 2012 Bertran released a book on code poetry:

Code can speak literature, logic, maths. It contains different layers of abstraction and it links them to the physical world of processors and memory chips. All these resources can contribute in expanding the boundaries of contemporary poetry by using code as a new language. Code to speak about life or death, love or hate. Code meant to be read, not run.”[8]

To honour the art form of poetry, it was important to him to make the book available in printed form. That’s what he has to say about the physicality of his book “Code Poems”:

"*It's been really exciting to design a book in which the code feels comfortable living in this physical medium, and poetry doesn't feel betrayed. Code literature might still sound strange but it feels right to me, to have around in our times.*"[9]

By making code available as a print, its performative function is taken away. Style and descriptiveness are the reasons to communicate it to others. If it was Bertrand’s goal to establish coding as a writing genre, he did that by printing it.

Is Code a Subjective Form of Communication?

“It doesn’t matter to the compiler how a piece of code is written, but it matters to the author and to the people the author has in mind.” (Croll, 2017)

In any form of writing, style is inherent to the text. Even though the instructions for the computer are the same, the way these instructions are expressed are not.

Code has its own rules (syntax) and meaning (semantics). Like novel writers or poets, coders also have their own style that includes strategies for optimizing the code being read

[8] Bertran, I. (2012). code {poems}.
[9] Solon, O. (2012). Designer publishes collection of developer-written “code poems.” Retrieved May 10, 2019, from https://www.wired.co.uk/article/code-poems-book

by a computer, and facilitating its understanding through visual organization and comments for other coders.

Apart from style, the intention for writing a text -not specifically code - has to be questioned. The reason why a text was written is a subjective one, no matter how objective its content may appear.

“Writing code is never automatic, value-neutral and disconnected from the meanings of the words in this world. Even if the computer interprets value-free, code is never value free.” [10]

Who writes code and why? Which problems should be solved by code? The market for software is driven by solutions for companies to handle their data traffic. Those who write the code are predominantly men who were trained at universities or by online courses to develop commercial software. A main goal for these programmers is optimization and profitability.

This excludes a big part of a society who instead of actively coding - and thereby creating solutions for their specific problems - become consumers of software products.

While coding literacy is praised as a new moral good and way of serving society, it is exclusive for a small group of people who have access to this knowledge.[11]

A big effort against programming knowledge as a privilege within society is the work of the Chaos Computer Club (CCC). The organization was established in 1984 and since then aims to make digital technology openly accessible and communities profit from decentralized networks.

The way the CCC acts, is based on the “Hacker Codex”. One of its principles reads “In order to understand something, we take it apart.” It's a behaviour that aims to fight passive consumption. Fiona, Krankenbürger, one of the speakers at the Chaos Communication Congress (an annual event of the CCC) says: " In order to use technology for different needs and interests, we need different people." [12]

In the new coding-literate society exclusivity is undermined by constantly building on databases that were developed in non-inclusive, biased conditions. Machine learning is a branch of programing where machines are trained to make decisions based on decisions that were made first by humans.
Commercial software based on machine learning algorithms becomes ubiquitous and governs our lives. Humans are not free of values, free of bias - and the algorithms they train are neither.

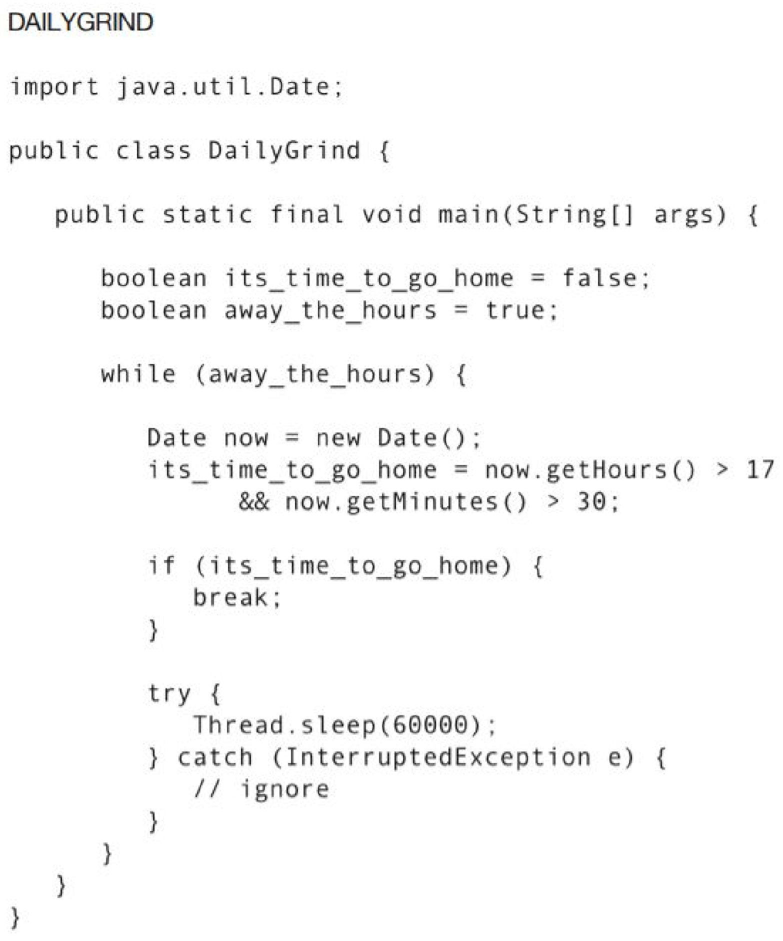
[10] Mataes, M., & Montford, N. (2005). A Box, Darkly: Obfuscation, Weird Languages, and Code Aesthetics. Proceedings of the 6th Digital Arts and Culture Conference, IT University of Copenhagen
[11] Vee, A. (2017). Coding Literacy: How Computer Programming is Changing Writing. In Software Studies.
[12] Krankenbürger, F. in Trostel, S. (Producer, Director). (2018). All Creatures Welcome. [Motion Picture]. Germany.

The need for reflection on how the technology is used and for which purpose becomes apparent in language. How do we talk about programming and the content of programs?

Naming conventions of common algorithmic tasks are questionable: “server”, “master”, “execute” and “kill thread” can be connected to blind obedience, colonialism, and law enforcement in authoritarian regimes.
These meanings became part of the software jargon (which does not make software developers by default supporters of any of these aforementioned practices) but show a lack of reflection when integrating new tools.

So leaving style and functionality aside, it is important to acknowledge that code is always a subjective form of communication.

1.4. Code Poetry



DAILYGRIND

```
import java.util.Date;

public class DailyGrind {

   public static final void main(String[] args) {

      boolean its_time_to_go_home = false;
      boolean away_the_hours = true;

      while (away_the_hours) {

         Date now = new Date();
         its_time_to_go_home = now.getHours() > 17
               && now.getMinutes() > 30;

         if (its_time_to_go_home) {
            break;
         }

         try {
            Thread.sleep(60000);
         } catch (InterruptedException e) {
            // ignore
         }
      }
   }
}
```

Fig 2: Code poem by Paul Illingworth written in the programming language Java from the book “code {poems}” by Ishac Bertran (2013)

At one point in history around 1950, programming a digital computer didn’t mean wiring anymore, but rather using words. The scientists who were working in this field at the time -

those writing the code as well as those who fed the instructions to the machine - were the first to experience language that was not meant to be heard or read by another person, but to be executed by a machine. (Vee, 2017)
Through their work this use of language also became part of the “computer culture”. How could this completely functional language be used when it got in contact with informal language or even poetry? Would this new form of writing become incorporated in literature?

Years later in 2014 Melissa Kagan, the initiator of the first Stanford Code Poetry Slams was asked what code poetry is:

> "It can be a piece of text that can be read as code and run as program, but also read as poetry. It can mean a human language poetry that has mathematical elements and codes in it, or even code that aims for elegant expression within severe constraints, like a haiku or a sonnet, or code that generates automatic poetry. Poems that are readable to humans and readable to computers perform a kind of cyborg double coding." (Kagan, 2014)

To sum it up, code poetry is the combination of what is expected from executable code on the one hand and abstract concepts formalised in language on the other. If these two come together in an interesting way, it can be considered code poetry. To form an opinion on whether something is code poetry - or even become a code poet - it’s required to have a sense of both.

The author of code poetry should be familiar with the language of computation and with the language to communicate with other sentient entities - both in a creative way.
There are several publications and events worldwide that emerged by bringing together people with aspiration as coders and poets. There are “Code Poetry Slams”, books of code poetry, “Code Obfuscation Contests”, esoteric programming languages and many other experimental formats in this intersection.

So what distinguishes a code poem from a classical poem?

A code poem, besides following the definition of a poem (which is the use of aesthetic and evocative qualities that contains multiple interpretations and thereby resonates differently with each reader) also fulfills functional qualities: The syntax that it uses stems from the language that is used to instruct computers.
Among the code poetry community it is argued if a code poem needs to be executable (able to be compiled without errors). Some code poetry competitions state this as a prerequisite for submissions.

If a code poem needs to be executable, it means that the poem has the ability to fail.
If a non-code poem works or fails is hard to determine. It is arguable that these concepts can be applied to a poem. It can not work or fail on any level except a personal one.

```
void run() {
  wonderer.lookfor(answers);
  wonderer.takesStepsAndClimbs(5, -3);
  wonderer.poses(question);
  wonderer.retrieves(emptyParadigm);
  intruder.attemptAnswer("Sometimes the
                      questions are complicated,
                      and the answers are simple");
}
```

Fig 3: Code poem "wonderer.wander" by the author

1.5. Language

Language is the method of human communication, either spoken or written, consisting of the use of words in a structured and conventional way.[13]

Each system of language helps with interpreting the world by describing and exchanging information about it. Languages work between individuals because of common rules of syntax and semantics known to all its speakers.

Syntactics (the direct content of symbols) and semantics (the meaning of symbols) together with pragmatics (how symbols are used) make up the study of semiotics, which is a general philosophical theory of signs and symbols that deals especially with their function in both artificially constructed and natural languages.[14] Semiotics also studies non-linguistic sign systems and thereby bridges from language to non-human-understandable code for machine instruction.

There are as many nuances to language as individuals who speak it - at the same time it can be considered as a unifier to merge all the different conclusions by means of translation and bring them into one form.

Functional, Poetic and Natural Language

Functional language is used when information needs to be transferred: The sender offers informative, structured language that is directed to a certain receiver or a possible target group of receivers.

Poetic language is used to train the mind on abstract concepts: information of artistic nature by itself is free of shape and purpose. It gets a concrete shape as soon as it is picked up by a person.

[13] language. 2019. In Lexico.com. Retrieved August 16, 2019, from https://www.lexico.com/en/definition/language
[14] semiotics. 2019. In Merriam-Webster.com. Retrieved September 18, 2019, from https://www.merriam-webster.com/dictionary/semiotics

Natural language is a language that has developed in use - in contrast to artificial language or computer code.[15] This language is different for each person and is part of the speaker's character.

Language Constructs Reality

According to Austrian philosopher Ludwig Wittgenstein, reality has the same relation to language as a subject to its depiction. It is needed to describe reality.

"Whereof one cannot speak, thereof one must be silent." (Wittgenstein, 1922)

In the beginning of his logical analysis of language he argued that It should be possible to reason about everything in the world there is by means of describing it with words. Furthermore, all sentences that describe the world can be deducted from elementary sentences. The truth value of these sentences can be determined through reasoning.[16]

His Tractatus Logico-Philosophicus which was published in 1922 consists of seven chapters and follow one continuous philosophical thought in which he tries to make sense and find the limits of the world. His goal was to be able to distinguish between sense and nonsense. [17]
Later, Wittgenstein rejected what he wrote in the Tractatus and called it "dogmatic"[18].
Natural, everyday language became the center point of his new theory. It let language and reality interact with each other. Every-day language is how we deal with reality.
The symbolized formal language of the Tractatus became something that helped to delimitate language and reality.

Imagination

All three forms of languages described in the beginning help us to understand and imagine the world. If we are familiar with the system of a certain language and if we are presented with information phrased within it, we can not not process this information.
Ideas take shape and spread through language. Symbols are translated into meaning.
Poetic language as well as logical fallacies in functional language let us imagine things that are not.

[15] natural language. 2019. In Lexico.com. Retrieved August 16, 2019, from https://www.lexico.com/en/definition/natural_language
[16] Biletzki, A., & Matar, A. (2018). Ludwig Wittgenstein. In The Stanford Encyclopedia of Philosophy (Summer 2018). Retrieved from https://plato.stanford.edu/archives/sum2018/entries/wittgenstein/
[17] Bach, J. (2018). The Ghost in the Machine An Artificial Intelligence Perspective on the Soul. 35c3. Retrieved from https://media.ccc.de/v/35c3-10030-the_ghost_in_the_machine
[18] Wittgenstein, L. in McGuinnes, B. F. (1979). Ludwig Wittgenstein and the Vienna Circle: Conversations Recorded by Friedrich Waismann. Oxford: Blackwell.

Examples:

- a hopeless person loses all hope
  (impossible, hopeless = without hope, no more hope can be lost)
- an empty island full of people
  (impossible, empty and full exclude each other)
- I went swimming in the desert
  (impossible, part of the definition of a desert is the lack of water, swimming can only be done in water)
- the doorbell rang and I let myself in
  (impossible, cause and effect are reversed)
- I am lying
  (impossible, paradox: the sentence is true and false at the same time)

These impossibilities expressed in functional language, obtain a poetic meaning. We can still imagine them even though they don't make sense.

1.6. Computation and Simulation

Computation is constructed mathematics. It's the "muscle work" in math where theorems that were inferred before are tested - it is also a way of proving a theorem. (Bach, 2018)
It is an executable set of rules formalized in language. Computation thereby plays the same role as the performer of a choreography: Executing or performing the rule set is a way of proving if they work in the reality that they were created for.

What is a mind?

From the perspective of computation and philosophy combined comes a strategy to find out what a mind is. Philosopher and computer scientist Joscha Bach argued in 2013 that computer science is the best discipline to tackle this question.. His arguments are the following:

> To test a theory on how the mind works, it needs to be simulated. Philosophy doesn't have the tools to do that. Neuroscience looks into the implementation of a concrete mind but not into how it works. Thereby it is a science of information processing. If Psychology would shift towards mental representations and away from behaviourism and experiments, it could have a key to the mind too. (Bach, 2013)

Bach concludes that only Computer Science

- has the tools to simulate information processing
- has the right level of abstraction to look at the mind in general and
- can work with mental representations.

Computational Thinking

"*Computer Science is not a science and its significance has little to do with computers.*" (MIT Press, 1996)

"*Programming is too useful to only use it in computer science.*" (Vee, 2017)

If we see programming not only as a way to instruct the computer but more as a way of thinking, we might come up with something like "Procedural Epistemology" [19]. Computers are thereby external thinking apparatuses here to help humans understand the world.

While computers have become valuable tools for constructing simulations of all possible realities - we make a theory, put it in a formal system, bring it to life by automatic computation and observe the result - it was the computational thinking that influenced philosophy and culture of the 20th century.

Computation is a neat tool for philosophical explorations:

- Abstract Classes and Generic Object: Object oriented programming attempts to create templates before using them as concrete objects. So it helps to think about the idea (a priori) instead of a concrete implementation of it.
- Infinity: Loops in theory can be repeated indefinitely and thereby help us question infinity on a theoretical level. By simulating infinity in an algorithm, the breaking of a system can be observed.
- Hacking one's own code: If we try to imagine ourselves based on a complex but working algorithm, we might make out a reward function: What is the motivation or sense of our lives and where is it encoded? Even if we don't find an answer to this question of all questions, we could still think about the following possibility: Can we hack our own "reward function"? Can we rewrite what gives meaning to our lives? Some people try to do that by meditation.
- Obfuscation: The act of deliberately making code less legible and/or understandable - while it still functions. It is the reverse act of demystifying a concept by breaking it down into elementary sentences. Starting from the early Wittgenstein´s premise that everything there is in the world can be described by language, obfuscation might be what constantly happens to natural language: individual use and natural development of language make its statements non-deterministic. Still their meaning and truth value can be inferred.
- The advent of computation in mathematics and the increased need to organize knowledge in formal systems in philosophy occurred in the first half of the 20th century and conditioned each other.

[19] Krebsbach, K. D. (n.d.). Computer Science: Not about Computers, Not Science. 4. Retrieved from https://www2.lawrence.edu/fast/krebsbak/Research/Publications/pdf/fecs15.pdf

Simulation

Simulation means to imitate a situation or process.[20] It is a tool of science as well as philosophy. Like a sandbox, it lets us experiment with our ideas about the world.

In order to understand something, we connect it with knowledge we already have. We build metaphors and scenarios to animate our theories. There is no bit of knowledge disconnected from any other bit of knowledge. It is either where something else is or where something else is not. Everything we can not know is by this definition disconnected from everything we know.
Sometimes theorems are discovered by filling the blank spots in our map of knowledge. There has always been ways to simulate knowledge and match them with reality.

Throughout history there were always ways of simulating theories: first it was language (first spoken, then written), then painting, theatre plays and games. Creating a simulation as close to reality as possible proves scientific understanding and skills.

Computation is the most advanced simulation apparatus known today.
Driven by input data about our world (like gravity, friction or geopolitical states) and computational power, it delivers better and better simulations in direct exchange with the growth of accumulated data and the knowledge we can extract from it.

[20]simulation. 2019. In Lexico.com. RetrievedSeptember 18, 2019, from https://www.lexico.com/en/definition/simulation

2. A History of Logic in Mathematics and Philosophy

Math, philosophy, computer science and sometimes language all try to describe reality with as few axioms as possible that don't contradict each other and work without exceptions or extensions. These fields try to deliver a complete representation of the world that ultimately leads to a better understanding and the exposure of inconsistencies and gaps.[21]

In order to complete the picture, problem may be approached from two sides: From one side we expand our knowledge. From the other we work on breaking the mysteries down to little digestible bits that we then try to match with axioms that were inferred and approved.

Logic is a discipline shared in all the aforementioned fields. Sometimes logical problems need to be attacked from both sides. When undecidable questions are asked in metamathematics, it needs philosophical help.

The philosophical approach - towards a strictly scientific question - might be similar to the one of art: critical and abstract thinking, thought experiments and possible renderings of the world go along with the application of logic.

The Relevance of Logic within this Exploration in three Steps

1. With logic we can find modes of agreement. Thereby, rules can be inferred of how to live together in a social system or how to reason on aspects of our lives by conversing with others or through contemplation. We can find out if something is true or false, right or wrong. It's a cognitive science that is branching into math, philosophy, language. It is also a solid basis for accumulating knowledge in the natural sciences.

2. One of the essential tools in propositional logic is the connection of two arguments into a complex one. A computer does exactly that and uses electricity to accelerate and automate the process.
So computers let us observe logic in action. They allow us to see if ideas about the world match how the world actually works. The number of questions that can be asked and worlds that can be created and observed is extremely high. Getting lost in all these possible worlds is a mental pleasure.

3. Logic is expressed and communicated through language. Natural language is faulty, and individually nuanced, expressive first and functional second. According to Wittgenstein language constructs reality, but which one exactly?
There were many attempts to create a deterministic language, free of paradoxes or interpretation, applicable to anything. If the rules of the deterministic language are executed by a computer, it should be possible to create a perfect simulation of the world.

And what about firing in the other direction: The rules of language can be subverted, a logical system can be played against itself, it's possible to break out of it.

[21] Hofstadter, D. R. (1980). Gödel, Escher, Bach: An Eternal Golden Braid (1st ed.).

This chapter will tell the story of one of the biggest metamathematical questions of the 20th century and how it was approached - but first some basics need to be established.

2.1. Propositional Logic

Since humans started to make sense of themselves - either alone or in exchange with others - one of the first tools needed was to know if they are right or wrong and - by that - what is true.

But how to do that? Different ideas of how the world might work lead to fundamental disagreements, some theories can only be speculated about but never proven.

So it might be a good idea to start at the very beginning: What can we know?
One idea that many big thinkers always came back to is a collection of things we know for sure and from there deduce and expand: Do we know that statement A is true and why it is true? If so, we can add it to the collection. And even if it is false: the antithesis can be added. Now this statement can be used and another thing can be concluded with the help of the truth in A.

This is **Propositional Logic**: A statement is proposed and declared either true or false by reasoning. **Atomic propositions** are those that can not be divided any further and more than one statements can be connected to another to form a **compound proposition**. The truth value of the compound proposition is then deducted from its parts.

The first documented use of propositional logic as a tool for deductive reasoning was by Aristotle who lived from 384 to 322 BC. The tool he developed was **Syllogisms**. (Bezhanishvili, Fussner, 2013)
A Syllogism consists of a major and a minor premise and a conclusion that follows from the truth content of the premises.

Example: Syllogism
```
Major Premise: Snakes are reptiles.        TRUE
Minor Premise: Reptiles don’t have fur.    TRUE
Conclusion: Snakes don’t have fur.         TRUE
```

Syllogisms - as part of Aristotle’s term logic - were limited because they don’t consider conclusions that can be deduced from more than two premises and at the same time require contextualization. Also they can not be connected to complex structures.

They were further developed by the middle eastern scholar Ibn Sina, also known as Avicenna who lived from 980 to 1037 and introduced tenses and temporal connotations (like never, always, at some times) and thereby created an early formal system using **Temporal Syllogisms**. Another work by Avicenna is the development of Syllogisms into “methods of agreements” that became an important contribution to the scientific method.

Another influential development of syllogisms was the introduction of **Logical Connectives** by the greek philosopher Chrysippus of Soli who lived from 279 to 206 BC.
A logical connective is an operator (symbol or word) that connects two or more propositions in order to infer a compound proposition - whereby its truth content is guaranteed by the proposition it is inferred from. These are the most common binary logical connectives.

```
=  IS       ⊻  XOR (exclusive OR)
¬  NOT      →  IF - THEN
⋀  AND      ← ONLY IF - THEN
⋁  OR
```

If now variables replace concrete statements, logical systems can be built.

Example: Logical Connectives
```
     Major Premise: Snakes are reptiles.        Snake **IS** Reptile
     Minor Premise: Reptiles don’t have fur.    Reptile **IS NOT**
     fury
     Conclusion: Snakes don’t have fur.         Snake **IS NOT** fury
     Snake = Reptile ⋀ Reptile ¬ fury → Snake ¬ fury
```

Let’s say this is not only true for snakes.
The same system of rules also applies to
- “Noon is during the day, during the day it is not dark, so at noon it is not dark.”
- “One is a positive integer, all positive integers are bigger than zero, so one is bigger than zero.”

The same rule system works in another context. In mathematics it is called an **Isomorphism**, but it is true for all fields where propositional logic is applied: Find out the basic principles of two different systems and they are isomorphic when their rule systems match. (Hofstadter, 1980)
From now on propositions can be handled without concrete examples. You can **abstract** a proposition and then deduce where else it can be applied. You work with symbols that stand for concrete things.

Example: Logical Connectives
```
     Major Premise: Snakes are reptiles.        S IS R
     Minor Premise: Reptiles don’t have fur.    R IS NOT F
     Conclusion: Snakes don’t have fur.         S IS NOT F

     Replace S with N for Noon, R with D for Day, F with DA Dark
     Major Premise: Noon is part of the day.    N IS D
     Minor Premise: During the day it is not dark. D IS NOT DA
     Conclusion: During noon it is not dark.    N IS NOT DA
```

```
IF S IS R AND R IS NOT F THEN S IS NOT F
S = R ⋀ R ¬ F → S ¬ F
```

But what if you want to describe not only what an object is but also what properties it has and which ones it doesn't have? And still describe a general rule that is not bound to a specific case?

With all of the above principles laid out and this question in mind we can proceed to **Predicate Logic**, a subcategory of propositional logic. While Propositional Logic treats whole statements, Predicate Logic distinguishes between objects and their properties (which are called predicates). It can be described as a collection of formal systems used in math, philosophy, linguistics and computer science.

To create a proper abstract statement within predicate logic we need **Subjects, Predicates, Quantifiers** and **Logical Connectives** to describe relations in general and then apply them to the specific. [22]

Logical Connectives	Quantifiers
`=  IS`	`∀  = for all`
`¬  NOT`	`∃ = exists (at least one)`
`⋀  AND`	
`⋁  OR`	`x,y,z` … Subjects
`⊻  XOR (exclusive OR)`	
`→  IF - THEN`	`P(), Q(), R()` … Predicate
`← ONLY IF - THEN`	

Example: Predicate Logic

```
Major Premise:  Snakes are reptiles.
Minor Premise:  Reptiles don't have fur.
Conclusion:     Snakes don't have fur.

Two Predicates: Reptile and has Fur
R(x) something is a Reptile
F(x) something has fur

One Constant: s .. a snake
One Variable: x .. something

The domain is the set of all things D.
Snakes are reptiles.       R(s)
```

[22] Waner, S., & Costenoble, S. (1996). Predicate Calculus. Retrieved from Introduction to Logic website: https://www.zweigmedia.com/RealWorld/logic/logic7.html

```
Reptiles don't have fur.   ∀x(R(x) → ¬F(x))
Snakes don't have fur.     ¬F(s)

∀x,s∈D
R(s) ∧ ∀x(R(x) → ¬F(x)) → ¬F(s)

x and s are elements of the same category, so everything
that applies to x also applies to s.
```

This example shows a system of **First Order Predicate Logic** where concrete examples are replaced by a set of abstract rules. But what if you want to formulate something that happens one level on top of that?

When one set of abstract rules becomes a subset of another set of abstract rules it is called **Second-Order-Logic**. But this is not where it stops: conceptually, any level of **Higher-Order Logic is possible**.[23]

Example: Second Order Predicate Logic

```
Major Premise:  The set S is a subset of the set R.
Minor Premise:  No element or subset of R is F.
Conclusion:     No element of the set S is F.

∀x,s∈D
∀s(R(s) )∧ ∀x(R(x) → ¬F(x)) → ∀s(¬F(s))

∀S(S∈R)
∀R(R(x) → ¬F(x)
```

For all possible sets that don't have a certain characteristic, all members of their subsets don't have that characteristic either.

There is at least one set within all sets that does not have a certain characteristic. Which means that all subsets of this set don't have this characteristic either.

```
∀S∈R
The set of all snakes is a subset of the set of all reptiles.
∀R → ¬F(R)
∀R(R(x) → ¬F(x))
∀S(S(x) → ¬F(x))

The set of all reptiles is not part of the sets of all things
that are fury. So the set of all snakes can not be fury.
```

[23] Leivant, D. (1994). Higher Order Logic.

One step back to propositional logic: a proposition is a predicate with no arguments.

One step forward: Now that we are familiar with logical reasoning, the next chapter will show how it was put into practice by great thinkers of math and philosophy. One big argument that took place throughout the first half of the 20th century showed why Higher-Order-Logic is not reducible to First-Order-Logic or rather: Why it is not possible to apply the same reasoning to individual elements and sets of elements - even if a set contains only one element.

2.2. The Foundations of Mathematics

Ar the end of the 19th century, the mathematical tradition had become rich and diverse. Mathematicians started to engage in a global exchange and it was hard for them to keep up with all ongoing developments. At this time, Henri Poincaré was the last mathematician considered a specialists in all subcategories of the field.

The vastness of mathematical research lead to different modes of proving axioms. Sometimes these modes were inconsistent and lead to different ways of proving theorems that were sometimes contradictory.

Today, science is more used to ambiguous results: quantum theory, infinity and non-euclidean geometry can be explained in different ways that contradict each other. Around 1900, scientists had to learn not to defend one solution like a dogma. (Bach, 2013)

A standard language in mathematics that unified all ways of logical reasoning even the possibly contradictory ones became a necessity. The next step then would be to define the form of logic used to decide what is true for each of them.
In 1897 the international congress of mathematicians took place for the first time in Zürich where Poincaré held the opening speech. This congress is still ongoing and at the time when this thesis was written, the latest edition happened in 2018 in Rio de Janeiro.

Philosophers were more trained in the art of answering undecidable polar questions. That's why especially the philosophers among the mathematicians community at the time occupied themselves with the substantial quest of a standardized reasoning in math.

There was one mathematician and “community manager” (Bach, 2018) who placed the call for action: In 1900 at the second congress in Paris David Hilbert asked all of his colleagues to solve metamathematics. He formulized 23 problems that would later be called “Hilbert's Program”. These problems were the undecidable ones that math and philosophy - for example by defining a standardized logical language.[24]

Unifying all mathematical principles means applying one logic that is able to explain everything - at least everything that can be expressed with language and observed in nature. Which language could be used to make the right logical conclusions? Natural languages are not deterministic, so a new form of expression had to be found.

[24] Launay, M. (2016). It All Adds Up: The Story of People and Mathematics.

2.3. Formals Systems

A formal system is used to infer theorems from axioms according to a set of rules. It may represent a well-defined system of abstract thought and may be expressed by a formal language. Unlike natural language it doesn't allow ambiguities and is constructed by premediation.

A formal language - like predicate logic - consists of function definitions, variables and the application of the function as a sentence or statement. Each statement should be inferred from another one that was concluded according to that which was established as part of the system.

Developing the ultimate formal system became a prominent task among philosophers and mathematicians at the time. One of the first endeavors was the "Begriffsschrift" by Gottlob Frege (1879) which was followed by "Principia Mathematica" (1913) by Bertrand Russell and Albert N. Whitehead. Then Ludwig Wittgenstein, a student of Russell published the "Tractatus Logico-Philosophicus" (1920). What they all tried to do was answer the question of determinism: if everything that exists is part of the same system, based on the same logic, there is no room for chance. [25]

2.4. "Begriffsschrift" by Gottlob Frege

The first formal system considered as such was a book on logic by the german mathematician Gottlob Frege called "Begriffsschrift" (roughly translated to "concept-script") published in 1879. Frege defines his goal to be the construction of an ideal language and also claims that his predecessor Gottfried Leibnitz failed in a similar attempt - which he thinks to be idealistic but possible.

Frege went on to employ his logical calculus in his research on the foundations of mathematics, and it went on to be carried out over the next quarter century.

A major part of the "Begriffsschrift" was the integration of set theory into the formal system. The set theory was developed by Georg Cantor. It is a branch of mathematics that deals with collections of objects. (Bezhanishvili, Fussner, 2013)

When the English mathematician Bertrand Russell got interested in Frege's work he discovered a major inconsistency which became known as "Russell's Paradox".
In 1902 he wrote to Frege that by all the rules established in the Begriffsschrift, one axiomatic question could not be answered. (Cryan, 2016)

For better understanding: Imagine a set of numbers {1, 2 3}. It contains the subsets {}, {1}, {2}, {3]}, {1, 2}, {2, 3} as well as {1, 3}. But is {1,2, 3} a subset as well? (Leivant, 1994)

[25] Maurer, E., Schürgers, N. J., Demmerling, C., & Gönner, G. (2015). Metzler Philosophen-Lexikon.

Let's say there is a set of all sets that contain themselves (R) and a set of sets that do not (!R). We can decide which nature {1,2,3} is of.
What about !R? Does the set of sets that don't contain themselves contain itself? If it does it does not and if it doesn´t it does. (Bach, 2018)

```
∃x∀y(x∈y → x ∉ x)
y∈y → y ∉ y
```

Frege had to admit that the Begriffsschrift did not deliver a solution to this problem. From then on the set theory described in his work was called “naive set theory”.

2.5. “Principia Mathematica” by Bertrand Russell and Albert N. Whitehead

Philosopher and mathematician Bertrand Russell and his colleague Albert N.Whitehead started to work in Oxford on a new formal system that should solve all inconsistencies in mathematics while keeping in mind what they learned from Frege's failure.
It took them 30 years to publish all three volumes of Principia Mathematica from 1910 to 1930.

Written as a defense of logicism (the thesis that mathematics is in some significant sense reducible to logic), the book was instrumental in developing and popularizing modern mathematical logic.[26]

They wanted to ban the paradoxes of naive set theory by developing the type theory. This theory in the end just forbade the possibility of sets that don't contain themselves. By that they grounding arithmetic, but did not fully base it on logic.

Still it seemed that a standardized method of reasoning was uncovered because nothing in the Principia seemed to contradict each other.

The books were very comprehensive and comprised more than just set theory. They tried to include all branches of math and it took the authors 362 pages just to be sure all axioms were proven that lead to 1 + 1 = 2. (Launay, 2016)

2.6. The Incompleteness Theorem by Kurt Gödel

Soon after, the whole endeavor of comprising all mathematics - and metamathematics - in one formal system was rendered impossible by the German mathematician Kurt Gödel.

In his publication “On Formally Undecidable Propositions of Principia Mathematica and Related Systems“ (1931) he did not only prove that the Principia was inconsistent, but did

[26] Linsky, B. (2019). Principia Methematica. In The Stanford Encyclopedia of Philosophy. Retrieved from https://plato.stanford.edu/archives/fall2019/entries/principia-mathematica/

that on a level that all further attempts of universal formal systems would fail. For the research on artificial intelligence it meant that an artificial mind could never be the same as a natural mind.
So how did he reason?

"No formal consistent system of math can ever contain all possible mathematical truths, because it can not prove some truths about itself." (Gödel, 1931)

Similar to a set of {1} not being the same as 1, the conclusions about a definition of something, can never be part of the defined thing. There is always an inherent difference in the signified and the signifier.

Even a perfect simulation of the world is not the same as the world because the reason for its existence is different. We don't know why the world exists, but we would know why the perfect simulation exists: In order to find out why the world exists. (Hofstadter, 1980)

2.7. Computational Machines

The mathematicians and engineers Alan Turing and Alonso Church developed computers independent from each other but at the same in the 1930's. It fit the time when formal systems in mathematics became so important and needed to be proven effectively. Church and Turing did not only want to create rule systems, but calculate to which results they might lead.

Their goal was building simulations that could run automatically. Then they would see if a formal system would lead to contradictions or not.

The modern computer was first mentioned in Turing's paper "On Computable Numbers" in 1936. He proved that such a machine is capable of computing anything by executing instructions (program) allowing the machine to be programmable. The fundamental concept of Turing's design is the stored program, where all the instructions for computing are kept.[27]

According to the Computability Thesis or Church-Turing thesis, all computers have the same power and can compute the same theorems - if resource limitations are ignored. The question of computability is still concerned with the question of what is computable and what is not.

In logic and computation the notion of "Turing-completeness" became an important defining factor. A programming language by definition needs to be Turing-complete: allow all algorithmic functionalities that can be computed.

One undecidable question in here is the "Halting Problem": It is undecidable by a computer if a program ever halts or loops.

[27] Church, A., & Turing, A. M. (1937). On Computable Numbers, with an Application to the Entscheidungsproblem. The Journal of Symbolic Logic, 2(1), 42. https://doi.org/10.2307/2268810

The question of what is computable exemplified by the Halting Problem lead back to Gödel: There is no possible algorithm that can reason about all possible algorithms. (Hofstadter, 1980)

Turing proved that his "universal computing machine" would be capable of performing any conceivable mathematical computation if it were representable as an algorithm. And the Halting Problem is not decidable algorithmically.

2.8. Ludwig Wittgenstein: A Deterministic Language

This chapter is again about language. Ludwig Wittgenstein who was a student of Russell started on the premise that *language constructs reality - and logic shows the border of language*.[28]

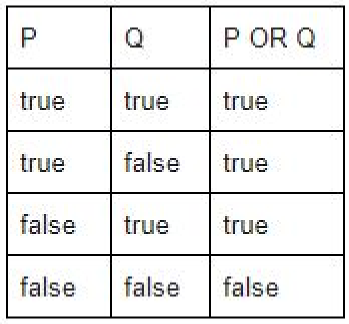
His major work “Tractatus Logico-Philosophicus” (1920) is the attempt to create a deterministic picture of the world by first reverse-engineering language and then rebuilding it from elementary sentences. This idea preceded computational machines but had the same motivation: Create axioms about our world and then logically deduce everything that is true. Upon reviewing his work from today’s perspective, it seems that this thought experiment was rather referring to a “programming language” than to natural languages. (Bach, 2018)

Wittgenstein‘s logic closes the circle to propositional logic: elementary sentences can either be true or false and can be combined to form complex sentences. It can be determined if the complex sentence was true or false exclusively by examining the boolean value of the elementary sentences that the complex sentence consists of. [29]

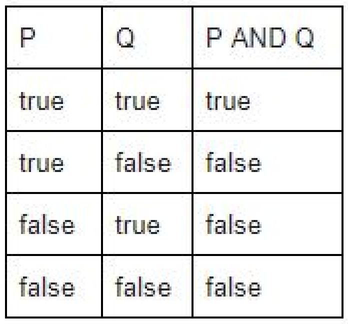
Wittgenstein attempts this with a truth table. While Bertrand Russell was developing a truth table for the publications “Principia Mathematica” to check mathematical symbols against each other, Wittgenstein developed this matrix for material implications.

A truth table serves to bring axioms in relation to each other. This is done by the use of logical connectives like NOT, AND, OR, NOR. They are known today in boolean algebra and as the basic components of digital computers or“logical gatters.

P	NOT P
true	false
false	true



P	Q	P OR Q
true	true	true
true	false	true
false	true	true
false	false	false



P	Q	P AND Q
true	true	true
true	false	false
false	true	false
false	false	false

[28] Maurer, E., Schürgers, N. J., Demmerling, C., & Gönner, G. (2015). Metzler Philosophen-Lexikon. Retrieved from https://www.metzlerverlag.de/suche/#
[29] Wittgenstein, L., Russell, B., & Ogden, C. K. (1922). Tractatus Logico-Philosophicus.

2.9. Boolean Logic

Boolean logic is the paradigm on which a computational machine functions. It deals with the combination of two states - either On or OFF, TRUE or FALSE, zero or one and allows data structures like IF/ELSE or logical connectives like AND/OR. Together with some memory space where the bit shifting takes place, Boolean logic and its data structures are all that a turing-complete computer requires. (Cryan, 2016)

There is an interesting isomorphism that explains why boolean logic works in theory and also in electronics: An electric signal and the absence of an electrical signal (ON and OFF states) within computational machines are processed in the same way as propositional knowledge does with TRUE and FALSE statements.

Even though computers came after Wittgenstein, he already knew that a NAND-gate (the basic component of each computer that processes one signal at a time) is sufficient to explain boolean algebra and thereby the functionality of the truth table. (Bach, 2018)

In electronics, a logic gate is an idealized or physical device implementing a Boolean function; that is, it performs a logical operation on one or more binary inputs and produces a single binary output. (Bezhanishvili, Fussner, 2013)

2.10. Programming Languages

Around 1950 natural language and computation met to form programming languages. Some notable programming languages were LISP (based on the Lambda-Calculus - a modular approach to formal systems developed by Alonso Church), C++ (a language on the machine level) and the more conceptual Object Oriented Programming. (Vee, 2017)

While Wittgenstein started out with natural language, reverse engineering it and creating a deterministic language, programming languages came from the other end: From boolean operations, mathematical axioms and propositional logic, languages were created that made it easier for humans to write algorithms.

2.11. The Artificial Mind

As already mentioned, computer science might be a good way of approaching how the mind works. Constructed mathematics allow us to calculate as well as simulate the mind. Even though we would not reach this goal, with every attempt we would find out what our mind is not. Without computation, complex simulations would be impossible and rules of how the mind works would be even harder to prove.

A conclusion of this history of logic within computation is thereby the attempt of finding answers to very profound questions about life by utilizing the power of external, constructed “minds”.

These endeavors - despite Gödel‘s incompleteness theorem are following the goal of finally having a machine that can’t be distinguished from a human thinking apparatus.

All the research on logic mentioned in the chapters above - from Frege to the development of programming languages - which resulted in evermore complex formal systems shaped the way we apply logic today.
Boolean logic is strongly incorporated in everyday reasoning. At the same time it gives a certain comfort that we know that computers can only reason based on boolean logic and we humans can do more to that.

We are those that create the computers, no computer ever created us.[30]
And still, a new shift can be observed: Latest machine learning algorithms make it impossible for us to fully understand how the logic behind them works.
They appear as if instead of clearly following path A or path B, they lean towards one path and don't abandon the other one completely. This behaviour makes artificial brains more human-like.

[30] Stocker, G. (personal communication, September 4, 2017)

3. Artistic and Scientific Methods

This chapter covers a personal interest in artistic and scientific methods throughout code and poetry. In theoretical research as well in philosophy, literature and programming, I encountered the same principles that help subvert logic, enable creative writing as well as creative interpretation of functional language which always surprise me with their results.

What they all have in common is their nature of playing outside the rules that constitute them. I think whoever uses them - a poet, a logician, a programmer or a writer - needs to understand their discipline thoroughly first, so they can eventually subvert the rules.

3.1. Infinity

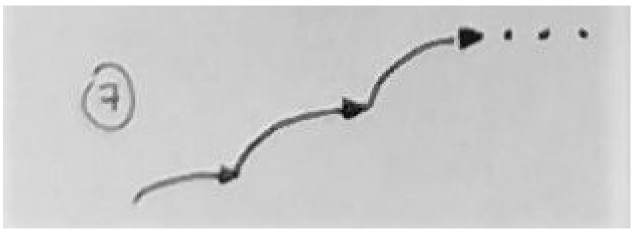


Fig 4.1: illustration of infinity by the author

Humans can not imagine infinity. So far we found out that there are infinities of different sizes - arguably the proof systems that speak for and against the existence of different sizes of infinity are not consistent which makes the issue again undecidable.

It is a topic that sparks constant fascination and fiction is full of stories about infinity. Many of them aim to give us an understanding of it and revolve around the following question: How does a conscious mind react if it was exposed to infinity?

By using computation we can simulate infinity as a program executed on a machine. This machine first of all does not mind repetition like humans do and just runs - meaning that the actively observing mind is missing. But even without that: It will run out of resources at some point. Power supply or memory, at some point the program breaks.

If the observer tries to think themselves into the simulation, it can be experienced like the break of (substituted) reality.

This again brings us closer the assumption that reality can not be observed in our reality.

Example: Java-based computer program that when executed, terminates because it attempts infinitely many increases of runtime memory.

```
ArrayList<Long> freeMemory = new ArrayList<Long>();
Long memory = 0;
```

```
void draw(){
      Memory = Runtime.getRuntime().freeMemory();
      freeMemory.add(memory);
      freeMemory.addAll(freeMemory);
}
OutOFMemoryError: Could not run the sketch.
```

3.2. Self-Reference

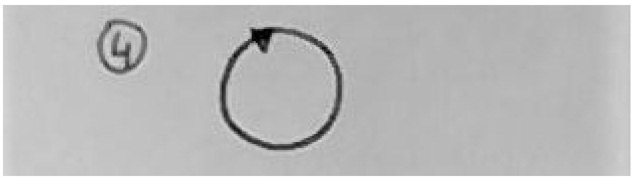


Fig 4.2: illustration of self-reference by the author

In order to truly understand how something works, find out about its nature, its rules, its motivations and use them against itself. A sentence about a sentence is the best description of a sentence. “I” is the best way if a person wants to refer to themselves. If a public city map is a good city map, you can find its own position on the map, even though you want to find all the other places but the map.

Self-reference are not paradoxes per se but might become contradictory. A purely self referential program does not crash, but also has no other purpose than pointing at itself in the memory.

One question that is interesting is how much self-reference is needed to describe something? It is a balance between too much content and nothingness. One could argue that a looped image that consists only of one image is a perfect self-reference - but is it? Ultimately the concept of a loop is lost.

3.3. Recursion

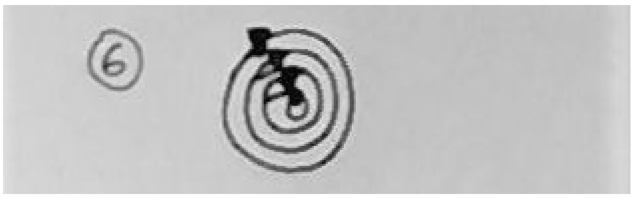


Fig 4.3: illustration of a recursion by the author

By definition, a recursion is the same as infinity. It is a certain type of infinity that is paired with self-reference. Recursion converges, infinity diverges. A recursion makes the space infinitely narrow within which it exists. Recursion is a common principle used in software development - and needs to be stopped from becoming an infinite self-reference. It is a predefined operation that references itself.

Example: Java-based computer program that calls a function which is calling itself indefinitely due to the lack of a termination condition.

```
void setup(){
      call();
}

void call(){
      call();
}
StackOverflowError: Could not run sketch.
```

3.4. Repetition

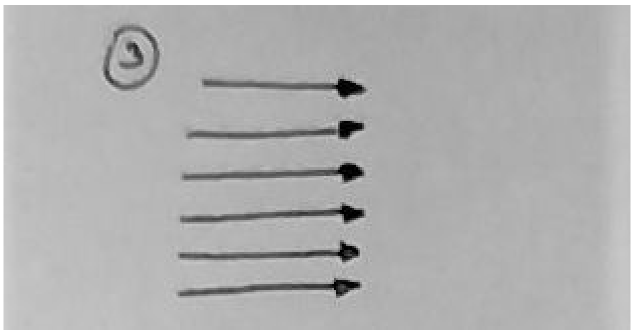


Fig 4.4: illustration of repetitions by the author

A repetitive action is easy to describe, easy to hand over to a computer. It's what computers are really good at: repeating something in the same way over and over again. In the end, the definition of an algorithm is a process, that when repeated with the same input always leads to the same output.

Repetition as practice leads to quite the opposite result: The more something is repeated, the more somebody learns from repetition. The content and meaning of the repeated act changes over time. We cannot not change our attitude towards the repeated act. We get bored, bettering it or changing it deliberately.

If a human repeats something, it will create an extra value outside of the repetition.

3.5. Paradox

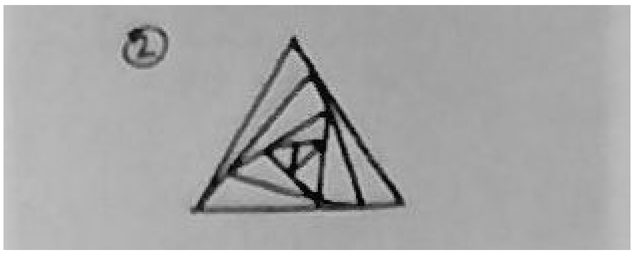


Fig 4.5: illustration of a paradox by the author

Playing the rules of logic against itself. When creating a formal system, without question the task is to make it logically coherent. And yet it is so easy to use the rules of the system to create logically impossible statements. It does not matter in which discipline: language, programming, philosophy - at some point we will halt at a logical fallacy.

So better to use it as a chance: It is so easy to create something that it is not, let's use it as a creative tool of expression, to add nuance to language and do what we are much better in computers: determining if the glass is half full or half empty.

Example: Java-based computer program that is designed to negate the truth value of its only variable whenever the variable is not negated.

```
boolean a= true;

void draw(){
      if(a==a){
            a!=a;
            print(a);
      }
}

Console:
truefalsetruefalsetruefalsetruefalsetruefalsetruefalsetruefalsetr
uefalsetruefalsetruefalsetruefalsetruefalsetruefalsetruefalsetrue
falsetruefalsetruefalsetruefalsetruefalsetruefalsetruefalsetruefa
lsetruefalsetruefalsetruefalsetruefalsetruefalsetruefalsetruefals
etruefalsetruefalsetruefalsetruefalsetruefalsetruefalsetruefalset
ruefalsetruefalsetruefalsetruefalsetruefalse...
```

3.6. Nothingness

Fig 4.6: illustration of nothingness by the author

The absence of something can only be observed if it is missing. If there is something that it can be (at least conceptually) be taken away from.

4. Artistic References

This is a collection of artistic references that encapsulate the concepts described above and also serve as an inspiration for the practical thesis projects “FURTHER, FURTHER, FURTHER” and “For while { Exception println #update let !doctype @type } ;”which are described in chapter 5 and 6.

The referenced works are of the same abstract, conceptual and out-of-the-box nature that is described in the theoretical chapters. Here, they come to life.

Throughout media like short-film, interactive installation, a non-fiction how-to book, a performance, a contest, a piece of code – they often explore and question their own rules – the rules of the medium, the rules of the message that is contained.

They try to step out of themselves: the short-film includes the director, the code a way to alter itself, the installation its own creation, the performance includes the institution that is housing it.

Some of them try to translate rules from one medium to another, some use repetition of a process to create more than the sum of their parts. The translation itself also adds extra value.

While exploring all of these projects, the artists’ plays an important role and becomes part of the project: how much do they reflect on the meaning of their work?

4.1. “Variable” by Maja Burja and Urška Aplinc

Disclaimer: Maja Burja is interested in how a concept or idea might change through individual interpretation – I compare it to the game “whisper down the lane”. I want to embrace this interest and give my own interpretation of her work.

“I link motivation to intent, while this work was perhaps shaped more by restrictions and necessities, fueled by chance and transient construction of meaning.” (Aplinc, 2019)

“Variable” is a performance by Slovenian artists Maja Burja and Urška Aplinc. It is strongly process based and not documented for anybody who was not part of it. The artists decided not to present their work in order to avoid influencing further iterations. It was performed several times and described by art critiques, of which the following conditions can be derived: a group of about 10 participants gather in an empty space, contextualized as an art institution. They define their goal themselves and it might lead to the collective creation of an artwork by theorizing about the possibilities of one.

“The title of the work is Variable and so is its content,” says Maja Burja. She and her colleague try not to influence the group of participants in their discussions and collective art creation. They offer them texts and images from previous iterations of the event.
Variable also means that the collective process is unavoidably influenced by external circumstances.

Burja and Aplinc don't offer any content or goal, their guidance stops when the event starts: the right setting is established, the participants are there and from that moment the artists stick to documentation. Repetition of the performance is part of the format: this way, the result of one event can be fed as input to the next one. A critique calls it autopoiesis: a system that recreates and maintains itself.
The “code of conduct” is then inferred parallel to the performance and it can’t be distinguished if the performance causes the rule system or the other way around.

“Thus in Variable the history of subjective, individual interpretations of a work of art become the work of art itself.” (Bojan Stefanović, 2016) The audience's response to the performance is its material.

Seen from a conceptual point of view, even the format of the performance is created during the performance. The art it results in is defined collectively as part of the process. It is a classic self-referential process that justifies its existence by its own creation.

At the same time, each iteration can be directly referred to and has a concrete manifestation. Each iteration has another narrative, composed by the participants. The concept of the performance might be self-referential, but its content is not.

The artists become observers and have the performance come to life through the setting, audience and the documentation that they can reuse.

4.2. “Hybris” by Arjan Brentjes

Fig 5: Still frame from the film “Hybris” showing actor Bart van der Schaaf and director Arjan Brentjes

In his six minute short film “Hybris”, Arjan Brentjes uses techniques of self-reference to demonstrate the possible implications of an eternal life. By executing the idea instead of describing it to the viewer, the understanding of it is more profound and immediate.

Plot

The story follows a dialogue set in a 1960s TV show where the interviewer asks his interlocutor the initial question, “*So, you really do believe that at some point we can live indefinitely?*”.
While maintaining a friendly conversation, they start to elaborate on the implications and consequences of eternal life.
The film is supported by repetitive background music which is mentioned towards the end when the expert says, “Perhaps eternity is just like a broken record. Playing the same part over and over again.” The interviewer responds by, “I hear.” Then the plot pauses for a few seconds to give both the characters and the viewer time to review their perception of the repetitive background music.
After that, the interview continues, “What was the initial question?” Within the same scene, the film resets itself when the interviewer asks the interviewee the initial question, *“So, you really do believe that at some point we can live indefinitely?”*.

Eternity is exemplified in two ways: an indefinitely small fraction of it is projected into a film, but it leaves the storyline open to go on forever. Also, this film is a loop: Something can be recognized as a loop, or repetition, if the beginning and the end are known and the beginning follows the end. If there is no breaking condition in sight, it is possible to imagine an indefinite repetition.

The following propositions are given about eternal life:

- **So, you really do believe that at some point we can live indefinitely?**
- Belief is not relevant for the possibility of eternal life, but science is.
- **And what will you be doing in your eternal life?**
- If science allows us to endlessly extend our lifespan, we might find out about the soul or fate.
- Endless life means either endless opportunities or the same fate over and over again.
- Humans are no longer produced in sequences. They should adapt and improve within their own life.
- The indefinitely living individual has no more need to produce offspring. Reproduction is redundant. Love and emotions lose their evolutionary function.
- The soul will be stripped down to its barest possible core - but will we then uncover a soul at all? It might have been an evolutionary necessary illusion.
- Assumed that happiness can only be found in moments of a mortal frame, and humans strive for happiness, they might voluntarily reject eternal life.
- The intrinsically destructive nature of happiness: humans especially enjoy the things that they could die from.
- Hybris: God intentionally diminished human's lifespan. Wouldn't humans be punished for undoing that?
  Christians believe that the accounts are settled in the afterlife and if humans never reach that, they can be free from God's will.
- Greek Mythology: Zeus wants to prevent the other gods from giving humans powers that are only for the gods. When Prometheus does it, he should be indefinitely punished.
- Endless life may mean endless agony.
- **Perhaps eternity is just like a broken record. Playing the same part over and over again.**
- **So, you really do believe that at some point we can live indefinitely?**

The questions and discoveries in **bold** are those that are directed to the viewer.

Further, we don't know the exact role of the "expert" within the frame of the film. Did he write a book? Is he a philosopher? He seems to just know. As he is portrayed by the director and writer Arjan Brentjes, it can be interpreted that Brentjes created the movie so he could be interviewed as himself – an entity from outside. By putting himself in it as a character, he can observe this character as part of the simulation of the eternal loop.

The interviewer is described as devoted to God. He is also stuck in the frame of the movie and has the opportunity to gain knowledge about something that is happening outside, but remains sceptical, as it is intrinsically impossible to prove something that does not exist inside his own story-world.[31]

[31] Brentjes, A. (2014). Hybris. Retrieved from https://vimeo.com/97523071

4.3. The Useless Machine - A Poetic Object

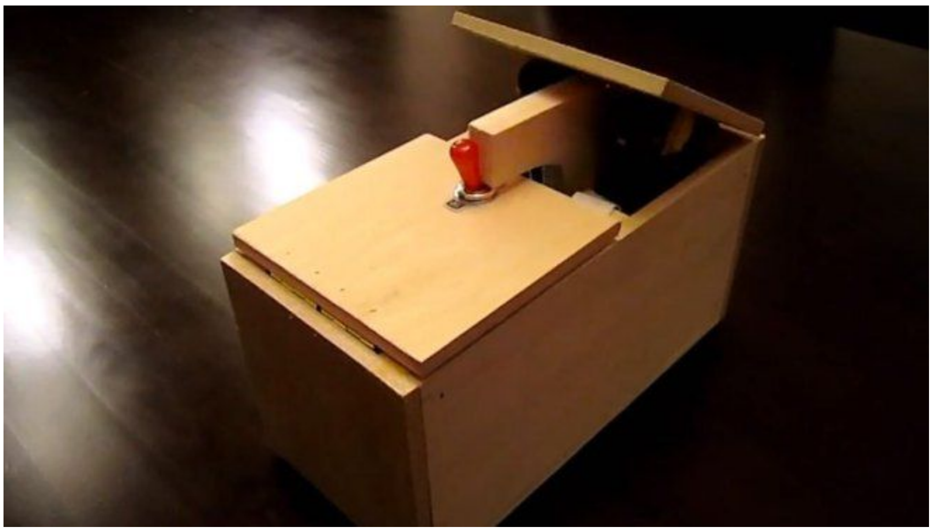


Fig 6: Bergstrom, K. (2010). DIY Useless Machine. Retrieved from https://vimeo.com/34453539

One of the inventors of the useless machine was Marvin Minsky, pioneer in the field of artificial intelligence. Actually, he called it “ultimate machine”, but the term “useless” prevailed. The purpose of the? “useless machine” is to switch itself off as soon as it is turned on.
On a meta-level, its purpose might be to make a philosophical point: what is the meaning of a machine? And what’s its purpose? Is it also a machine when it is turned off? What are the core elements necessary, so something can be called a machine? (source: The Technium)

According to Wikipedia, a machine is a mechanical structure that uses power to apply forces and control movement to perform an intended action.
From an existential point of view, it can be argued that it is not a machine, if no power is used. The “intended action” in this case is to only use power to cut power off. In conclusion, the machine uses power only so it is not a machine anymore.[32]

For this reason, Minsky called it “ultimate machine”, because it was able to break out of its own system: by refusing to perform any intended action but refusal. From the field of AI, it can be argued that this is the first and only machine that is able to refuse to perform. But it still is not an example of “free will”, as it was designed to turn itself off.
Also, a clear “behaviour” can be observed in the machine. It is a perfect mirror of the determination the user has to turn it on. The machine shows its own determination by turning itself off.

The useless machine is a generous example of self-referentiality, paradoxes and breaking out of a logical system.

[32] Wikipedia contributors. (2019, April 27). Useless machine. In *Wikipedia, The Free Encyclopedia*. Retrieved 22:39, September 18, 2019, from https://en.wikipedia.org/w/index.php?title=Useless_machine&oldid=894312051

- It is not connected to any process of mechanical production and echoes every input back as an output as soon as it receives it: electrical current applied through the “ON” switch is immediately cut. The intended action of a person to turn it on is immediately refused by reversing the act and putting the switch back in its original state.
- In this case, one could say that the machine is working very hard to fulfill its purpose of NOT being a machine. The harder one tries to switch it on, the harder it tries to switch itself off. By refusing to be a machine, it is the perfect example of a well performing machine.
- A machine is a system designed for a certain purpose. It can malfunction, given an unpredictable number of unpredictable influences. Still, it operates within a given frame , or as in the field of machine intelligence, a given “problem space”. Breaking out means bringing the machine on the same level as its creator – a level on which they don’t have any influence on it anymore.

4.4. Design Studio Mischer’Traxler

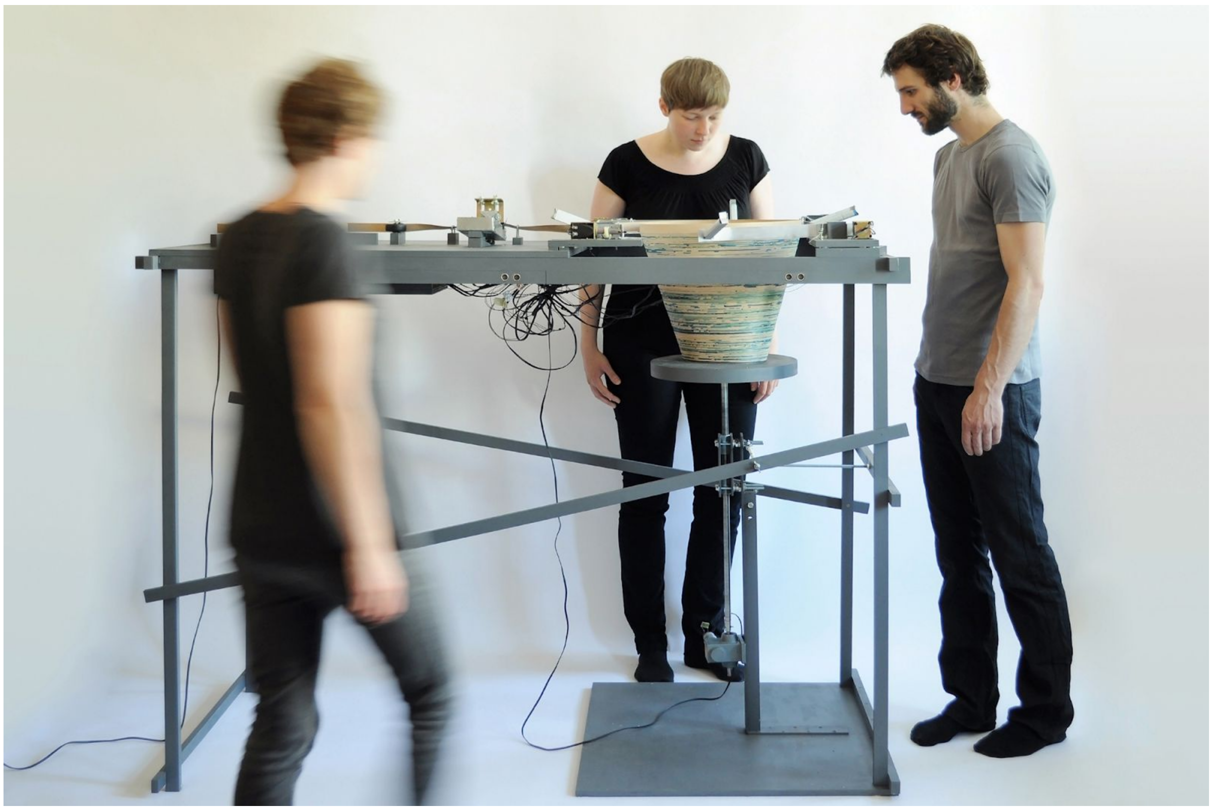


Fig 7: “Collective Works” (2011) by Studio Mischer’Traxler

Many of the design projects by the Viennese studio Mischer’Traxler are based on an initial thought that is applied to itself where products are little excerpts cut out of possibly endless processes. Especially the concept of self-reproduction is explored thoroughly: the trigger that starts a process is included in the process itself. Or: the purpose of an object’s existence is part of the object.
What does it mean when the trigger for an object, the reason why it exists, is part of the object? If there was no trigger, then the object would not exist. That makes sense. Without

reason, starting point or production, there will not be a product/the result will not be a product.

In these projects, the triggers are not only of a conceptual nature, not only the initial idea for the project to be made, but a design factor: the trigger manifests itself as a design pattern, a rule of production, the moment to finish one product and start a new one.
Sometimes, it is even visible how much the object "wants" to exist: stronger or weaker material, more or less effort in the production process – these show different signs of "interest" in the existence of that very product.

There are certain rules for the object to come into existence, to be produced and to be finished. These rules are then executed like an algorithm that allows input parameters and several breaking conditions.

Collective Works (2011)

The outcome of this project is a basket. This basket is the recording of the production process of the basket itself, including a representation of the people who watched it being made. The product is the record of its own creating, including the interest in it being made.

"Collective Works" is set up as an automatic basket manufacturing machine. There is a wooden veneer strip on a roll that is slowly pulled through glue and then around the base of a basket which is slowly moving down. The spinning and the downwards movement result in the basket. The manufacturing machine also includes sensors that notice if somebody is taking an interest in the installation. Only then the machine is moving and the basket is being made. Along the way of the veneer strip, there are four colored markers: their color is applied if more than one person is observing the basket being made. Each person means an additional color layer on the basket. Hence, the basket becomes a data visualization: the record (veneer strip moving and being glued) starts with one person, resulting in bits of no color. Up to five people can be recorded: the darkest strip means all four colors are applied. If one looks at the finished product, the height of the basket shows the time of the creation process and the pattern the fluctuation of observers.

Mischer'Traxler speak of a "production on interest":

"If nobody is interested in the project, it stops producing at all and the final object just does not get made. (Mischer, Traxler, 2011)

Considering the philosophical aspects of production, re-production and self-reference, an object that is the reason for its own existence is interesting to observe. Gödel's incompleteness theorem says that the instance of something can never be itself, but only a copy. That is, because it cannot include the reasons why it was made, or any information, one layer above the object.
However, this product seems to include its own reason of existence. The product is at least a well made attempt to counter-argue Gödel's theorem.

The following implications were incorporated into “FURTHER, FURTHER, FURTHER”:

- Something happens only if it is observed. This observation can be called the “desired factor” e.g. somebody is looking at it or hearing it.
- If the desired factor is not there, e.g. the observer leaves, the process stops working.
- The process is nurturing itself. It takes its outputs as inputs. The desired factor is the input of the process that then becomes an output and is fed into the mechanism again.
- By looking at the process of being made, the observer changes the mode of the product’s creation.

4.5. Exercises in Programming Style

In 2015, Portuguese programmer and educator Cristina Videira Lopes published the book “Exercises in Programming Style”, in which she compares 33 different styles of programming. All 33 examples are written in the same programming language (Python) and ultimately solve the same problem.

The task is an algorithm for term frequency including stop words. Term frequency means that a program is fed a piece of text and then outputs the most popular words in the text plus the number of times they appear. It can be designed to include stop words like “and” or “or”. [33]

Videira Lopes is an experienced programmer and argues that her software development students are never confronted with the concept of style in relation to the code they write during their studies.
She wants to emphasize that no piece of human writing can exist without style. Each programmer has their own unique one. It becomes apparent in a team when coders read each other's code and immediately recognize who wrote it.

Videira Lopes did not only collect 33 examples of the same algorithm and separate them in nine categories (historical, basic, function composition, objects and object interactions, reflection and metaprogramming, adversity, data-centric, concurrency, and interactivity), she also described their style: When did the style come up? Who uses it and when? In which scenario is the style advisable to use?

The online magazine Software Development Times added the book to the list of the “best programming books of the decade”. [34]
Their review reads: “... but perhaps Videira Lopes’ most brilliant decision was to write all of the programs in Python. The language is constant; the style is different.”

Lopes is interested in research on functionality of code language and discovered style as an important element in it. This way, she tied programming to creative writing and showed that it is in no way inferior to it.

---

[33] Videira Lopes, C. (2014). Exercises in Programming Style. CRC PRess.
[34] O’Brien, L. (2015). Code Watch: The best programming book of the decade. Retrieved September 18, 2019, from sdtimes.com website: https://sdtimes.com/books/code-watch-the-best-programming-book-of-the-decade/

The inspiration for her book came from the literature classic “Exercises in Style” (1947) by French writer Raymond Queneau. It is a collection of 99 retellings of the same short story, each written in a different style. Through that trick, Queneau made the book a standard work for aspiring writers and completely detached the content of a story from its style.[35]

It became clear what style could be: words, sentence lengths, moods or rhetorical devices that are all applied to the same story.
The book includes forewords by Umberto Ecco and the Italian writer Italo Calvino.
In his own books Calvino ingrains abstract ideas like nothingness as characters and narratives in his stories.[36] This approach became influential for the project “FURTHER, FURTHER, FURTHER”.

4.6. Techniques of Code Poetry

4.6.1. Subversion

Subversion refers to a process by which the values and principles of a system in place are contradicted or reversed. It is often used in a socio-political context but is also a common tool in activism and art.[37]

According to Annette Vee, code has two main functions: a descriptive function for human readers and a performative function for the machine that executes it. A programming language can be optimized for both: It can be intuitive to read and write for humans, following the principles of natural speech, making processes, cause and effect visible and understandable and allow different phrasings. It can also be efficient for the computer to process and optimize memory allocation, setting pointers, partitioning and threading.[38]

A subversive programming language would then be one that does neither: It is neither user-friendly nor efficient for the machine to process. It denies its descriptive and performative function.
Still both of them remain present: subversion does not mean that the functionality is lost completely:: description and performance are still present.

It is a game of finding the fine line: there is a lot of potential in creating these effects that oppose the widely recognized and desired functions, but still keep them at work. Especially in the field of esoteric programming languages there are many that try to stay within the definition’s boundaries but on the edge.

[35] Queneau, R., Eco, U., & Calvino, I. (1958). Exercises in Style.
[36] Calvino, I. (1962). The Nonexistent Knight.
[37]Wikipedia contributors. (2019, September 16). Subversion. In *Wikipedia, The Free Encyclopedia*. September 18, 2019, from https://en.wikipedia.org/w/index.php?title=Subversion
[38] Mühlbacher, J. R. (2009). Betriebssysteme. Grundlagen. Schriftenreihe Informatik.

4.6.2. Esoteric Programming Languages

An esoteric programming language (EPL) is a programming language designed to test the boundaries of computer programming language design, as a proof of concept, as software art, as a hacking interface to another language, or as a joke.[39]

To understand and classify EPL correctly, the concepts of Turing Machine and Turing-completeness need to be understood first: A programming language is Turing-complete or “computationally universal” if a program which is written in that language can be run on any “Turing Machine” and always lead to the same result. A Turing Machine is a machine able to do algorithmic computation and goes back to an abstract model which Alan Turing used as a basis for proving theorems about algorithms before electronic computers existed.

It is represented by a machine with a read-write head of an arbitrarily long length of tape. The tape can be shifted in both directions, each place on the tape holds one symbol that can be read by the head, the assigned response to the symbol can be executed and the symbol can be changed before shifting the tape again.
All known Turing-complete machines are able to perform the same calculations as this original, theoretical one through a set of statements, hence the “programming language”.[40]

By this definition, as long as the esoteric language is a programming language, it is Turing-complete. Still, there is a wide spectrum of how functional and descriptive these languages need to be. EPL’s play with the descriptive and performative function for the sake of artistic expressen, code poetry and finding smart ways of how the computer is processing the program - while often using bugs as features.

The Shakespeare Script

“All coders are playwrights and all computers are lousy actors.”[41]

Not only the “Choreographic Coding Lab” (an organization that brings coders and choreographers together) discovered that there are significant overlaps in instructing the computer to fulfill a task and performers. The written instructions are both called scripts, the playwright interacts with the performers, observes the process, changes it if necessary.

In 2001 the programmers Karl Hasselström and Jon Åslund took the expression “the computer performing a script” quite literally and created the Shakespeare Programming Language. It is an exercise in style transfer, where the style of a Shakespear play is applied

[39] programming language. 2019. In Merriam-Webster.com. Retrieved September 18, 2019, from https://www.merriam-webster.com/dictionary/programming%20language
[40] Turing Complete. 2014. In wiki.c2.com Retrieved September 18, 2019, from http://wiki.c2.com/?TuringComplete
[41] Unknown programmer, Retrieved from: http://www.quotationspage.com/quote/221.html

to the functionality of code. Both forfeit a good part of their essential criteria for the sake of creating something new: a Shakespear script that is executable.[42]

`Title: Two individuals found each other`	`The title is a comment`

`Romeo, a young man with a remarkable patience.`
`Juliet, a likewise young woman of remarkable grace.`
	`Declaration of 2 global integer variables: Romeo and Juliet.`
	`Variables are named after know Shakespear characters and need a description to be valid.`

`Scene 1: THE BEGINNING`	`A scene is a GOTO position.`
`[Enter Romeo and Juliet]`	`The variables that will be used in this scope.`

`Juliet:`
`You are the difference of your graceful sentimental big strong love and a stone. Speak your mind!`
	`The assignment of an Integer value to the variable Romeo. And the command to print it out.*`
	`1 + noun and 4 adverbs = 1x2x2x2x2 = 16`
	`1 - noun = -1`
	`Difference: 17`

`Romeo:`
`You are a beautiful shining lonely blooming flower and a shimmering beloved everlasting star as well as an annoyance. Speak your mind!`
	`Same for Juliet:`
	`1 + noun and 4 adverbs = 16`
	`1+ noun and 3 adverbs = 8`
	`1 - noun`
	`Addition: 23`

`Juliet:`
`Am I better than you?`
`If so let us proceed to scene 2.`
	`Comparison`
	`Juliet > Romeo = TRUE`

`Scene 2: THE END`	`Condition: If TRUE, GOTO Scene 2`
	`(There is no branch for FALSE, it’s not needed in this Example)`

<u>Syntactic Rules of the “Shakespeare-Script”</u>

- Blocks of code need to be addressed as “Scenes”
- Variables are introduced as characters native to a Shakespear play
- A variable declaration is made if the name of the character is followed by an (arbitrary) description
- The interaction with variables (assign values, change them,..) is achieved by another character speaking to it.
  This also means that in order to change a variable, there have to be at least two

[42] Hesselström, K., & Aslund, J. (2001). The Shakespeare Programming Language. Retrieved from http://shakespearelang.sourceforge.net/

characters on stage ("visible in the scope") who can talk to each other.
- Even though all variables hold numerical values, they can not be directly named. Instead nouns of positive (+1) and negative (-1) connotation are used. The value of them is increased by adding verbs to the noun.
- If a character asks a question, it tests another variable for a condition.
- It is possible for the compiler to jump in the code and it does so if a character mentions another scene - or other block of code.
- "Speak your mind!" is equal to printing a value to the console.

All these rules make the Shakespeare-Script Turing-Complete. And at the same time it leaves room for interpretation: The absence of numericals in the script let the author of the code decide about expressions that are either nice or insulting and therefore positive or negative. "Pig" can be -1, "flower" can be +1.

brainfuck: An Obfuscated Programming Language

brainfuck is an esoteric programming language developed in 1993 by Urban Müller and is notable for its extreme minimalism. It is Turing-complete and only consists of eight simple commands and an instruction pointer. [43]

The brainfuck compiler interprets and prints this program as "Hello World!".
```
++++++++[>++++[>++>+++>+++>+<<<<-]>+>+>->>+[<]<-]>>.>---.+++++++..
+++.>>.<-.<.+++.------.--------.>>+.>++.

Console: Hello World!
```

Digital Artist Daniel Temkien started an interview series with the creators of ESL´s in 2011 and wrote about brainfuck: it "refuses to ease the boundary between human expression and assembly code and thereby taking us on a ludicrous journey of logic." [44]

brainfuck exposes the inherent conflict between human thinking and computer logic. How "language-like" does a programming language have to be?
Programming and reading code within an obfuscated language becomes a form of art comparable to the performing an event score of the Fluxus movement.

4.6.3. Quine

Games of logic, creative writing and programming languages gave birth the quine, a self referential program: It is structured in a way to print its own code to the console. The name goes back to logician Willard van Orman Quine and was first mentioned by the author Douglas R. Hofstadter who exemplified Gödel's incompleteness theorem with it.

[43] Wikipedia contributors. (2019, September 14). Brainfuck. In *Wikipedia, The Free Encyclopedia*. Retrieved 00:00, September 19, 2019, from https://en.wikipedia.org/w/index.php?title=Brainfuck&oldid=915722954
[44] Temkin, D. (2011). Esoteric Codes. Retrieved from https://esoteric.codes

If a programming language is Turing-complete and able to output a string, it must be possible to write a Quine program in it. Quines can thereby be seen as proofs that the language works - proof by self-reference so to say.
If it applies its own rules to itself and it works, it is consistent.

Example: Lisp/Lambda Quine

```
           ((lambda (x)
                 (list x (list (quote quote) x)))
                (quote
                     (lambda (x)
                          (list x (list (quote quote) x)))))



Console:   ((lambda (x)
                 (list x (list (quote quote) x)))
                (quote
                     (lambda (x)
                          (list x (list (quote quote) x)))))
```

Example: One-line Python Quine

```
           s = 's = %r; print (s%%s)'; print (s%s)
Console:   s = 's = %r; print (s%%s)'; print (s%s)
```

Example: Java Quine

```
public class Main{public static void main(String[] args){String
f="public class Main{public static void main(String[]
args){String
f=%c%s%1$c;System.out.printf(f,34,f);}}";System.out.printf(f,34,f
);}}

Console:
public class Main{public static void main(String[] args){String
f="public class Main{public static void main(String[]
args){String
f=%c%s%1$c;System.out.printf(f,34,f);}}";System.out.printf(f,34,f
);}}
```

The 128-Language Quine Relay
Japanes programmer Yusuke Endoh created a 128 language “Ouroboros” quine. He used 128 different programming languages that each produce the Quine of another language and

thereby trigger the next program to compile. The first language is Ruby, it triggers a chain reaction until it starts again with the Ruby script.[45]

[45] Endoh, Y., 128-Language Quine Relay, GitHub repository, https://github.com/mame/quine-relay

5. Personal Project: FURTHER, FURTHER, FURTHER

Lecture-Performance at Gallery BB15, June 15, 2019

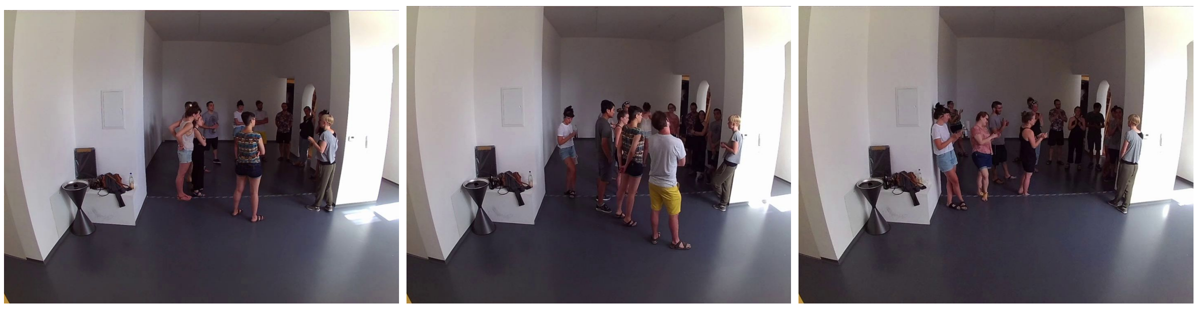


Fig 8: Documentation of "FURTHER, FURTHER, FURTHER" at Gallery BB15, June 15 2019

In order to put the research of this thesis into practice, an event on the **performative execution and poetic quality of code** was organized at the local gallery BB15 in Linz. It was intended to be an exercise of self-referentiality and loops. During the event and the subsequent discussion, it became apparent that the format allowed different interpretations, and that it was brought to life by the group who steered the questions posed by the event into surprising directions.

In the end, the event was described by participants as:

- a lecture performance
- an exercise of mechanical repetition and machine mimicry
- a guided tour about a guided tour
- an explanation of self-referentiality and loops by acting them out
- An exercise of self-reference
- the collective construction of art
- a space exploration
- the establishment of a rule system and a learning process of how to act within it
- location of information in time and space
- the result of a research on rule systems and conventions
- the performative execution of a computer program, comparable to a Fluxus score
- learning and sensory perception as an inherent characteristic of living organic beings
- building up a narrative through repetition of content
- learning by repetition and the? incapability of forgetting by will

5.1. Project Outline

The lecture performance was a one-hour guided tour through an empty exhibition space for a group of interested artists and theorists, followed by a discussion between the guide and the group. The participants knew before that they would take part in a tour, but were not informed about its content or its performative aspect.

The event began by the artist reading her own text on repetitive processes.

The tour itself had five stops and was repeated three times, but ended during the third round. Therefore, the audience walked two and a half rounds and listened to the same content two to three times. In sum, there were 13 stops. Each stop was assigned to one of the following five topics:
Self-reference and language, the event itself and rule systems, simulations and how to break out of a system, double meanings, physical presence.

The tour ended when the topic "how to break out of a system" was reached for the third time.

Invitation

20 people were invited and informed about the event, 13 participated. They received the following invitation:

FURTHER, FURTHER, FURTHER Opening, Guided Tour and Discussion, BB15, 4 - 6 pm This exhibition becomes the temporary center point of an ongoing exploration of how to get further. It is a guided journey through all the necessary steps in the endeavor to reach an end.

The invitation was misleading on purpose, because the expectation of the group was one material to shape the event. The expectation should be as general as possible: *a gallery space where there is art and somebody who knows about the art will give an introduction.*

5.2. Participants

The following 13 people attended the performance and stayed afterwards for a feedback round. Their input is quoted later in chapter 5.7. "Analysis"..

Marie Andreé-Pellerin Amir Bastan Loren Bergantini Davide Bevilacqua Sofia Braga Fabricio Lamoncha Katharina Edelmair	Gabriella Gordillo Hess Jeon Judith Maule Julia Nüßlein Onur Olgaç Antonio Zingaro

- All but three participants are affiliated with the Masters Course Interface Cultures: They either studied there when the performance took place(Bastan, Braga, Jeon, Nüßlein, Olgac, Zingaro), did so before (Bevilacqua, Lamoncha, Gordillo) or were visiting students (Bergantini).

- Most of them were invited because of there background in media art, their interest in conceptual art, their knowledge of programming and/or performance art.
- Gabriella Gordillo was involved in the planning of the event because of common interests in performance, logical concepts and programming.
- Fabricio Lamoncha is one of the three thesis supervisors, assistant professor and PhD candidate at Interface Cultures.
- Judit Maule is an art mediator at the OK Center for Contemporary Art in Linz. Andreé-Pellerin, like Bevilacqua, is responsible for the space BB15 and co-initiator of the event.
- Katharina Edelmair is an art and media theorist and art teacher

The participant are from 10 different countries (Canada, Iran, Brazil, Italy, Spain, Austria, Mexico, Korea, Germany, Turkey), 6 of them identify as male, 7 as female. Their age ranges between 25 and 40. At the time of the performance they all work practically or theoretically in the field of art and are based in Linz/Austria and were in contact with me during the research period of my thesis.

5.3. Content

Score

The performance followed a set of instructions: several repetitions of walking and talking preceded by the recitation of a text on repetition. This was the score written before the event took place:

1. I will start out by describing language and how it affects our thinking. The experiment here is imagining an artwork in the blank space.
2. Then we will continue to the next stop and talk about repetitive processes and how the act of repetition is helping to explain something that is repetitive. This also refers to the title of the piece which is "FURTHER FURTHER FURTHER"
3. Next, we talk about the history and the need for formal systems throughout art, mathematics and code but also the wish to break out of them.
4. As a 4th step I want to talk about double meaning. I start with my double role as a presenter and an artist. It’s about the juxtaposition describing and executing, being the container or the object contained.
5. Step 5 is about physicality, bodies and a reality check with the audience: Where are thex, what do they think about, what can they make of what they heard so far?
6. We are back at the beginning. Again I describe how language affects our thinking. Again I make the artwork appear in the "blank space".
7. We will continue to the place we visited before and I talk about repetitive processes again. How does the act of repetition help to explain something that is repetitive? We go FURTHER FURTHER FURTHER in a sense of going deeper.
8. Again I describe the history and rules of formal systems and the wish and fear to break out of them.
9. As a 9th step I want to emphasize again on double meanings. The text is the same, but the attention shifts now to the second meaning of all concepts mentioned before:

from presenter to artist, from describing to executing, from the container to becoming the contained object.

10. Step 10 is again a reality check with the audience, the same questions are asked.
11. For the third time the tour starts again by the same introduction and simulation of an artwork in space.
12. Again the act of repeating and self-reference are mentioned in the context of FURTHER, FURTHER, FURTHER.
13. The tour ends when the 3rd stop is reached and the topic of how to break out of a system is mentioned again.

Code

```
Topic language, further_further_further, break_system, double_meaning,
embodiment;

void setup() {
  setUpTopics();
  for (int round = 0; round<round+1; round++) {
    for (int stop = 0; stop<5; stop++) {
      Topic topic = null;
      switch(stop) {
      case 0:
        topic = language;
        break;
      case 1:
        topic = further_further_further;
        break;
      case 2:
        topic = break_system;
        break;
      case 3:
        topic = double_meaning;
        break;
      case 4:
        topic = embodiment;
        break;
      }
      walkToSpot(stop+1);
      talkAbout(topic);
      if (round==2&&topic==break_system)break;
    }
    if (round==2)break;
  }
}



void talkAbout(Topic t) {
  println("Talk about "+t.toString()+". ");
}

void walkToSpot(int s) {
  print("Walk to stop Nr. "+s+". ");
}
```

```
void setUpTopics() {
  language = new Topic("Language");
  further_further_further = new Topic("FURTHER, FURTHER, FURTHER");
  break_system = new Topic("Break System");
  double_meaning = new Topic("Double Meaning");
  embodiment = new Topic("Embodiment");
}

class Topic {
  String topic;

  Topic(String topic) {
    this.topic = topic;
  }

  @Override
  String toString(){
    return topic;
  }
}
```

Explanation

First, the five topics of the walk are declared: Language, Further Further Further, Break System, Double Meaning and Embodiment.

- Within the main-Function there is first a utility-Function for the topics called setUpTopics().
- Until this step the algorithm describes the “preparation” for the walk. Now the walk starts.
- Then, there are two nested loops: The outer one is constructed as an infinite loop: whenever it reaches the last round, another one is added.
- The inner one counts the 5 stops within each round. Each stop is assigned one topic. The assignment never changes.The counter of the inner loop is matched to the topic by switch-case Statement.
- As soon as the current topic is assigned, the group walks to the next spot by calling walkToSpot(spot+1).
- Then the information about the current topic is given by calling talkAbout(topic)
- Now comes the condition that lets us leave the loop: Only if we are in round 2and the current topic is break_system, we can leave the inner loop.
- In order to leave the outer (infinite) loop, again the question is asked if we are already in round 2.
- The program is over, we broke out of the two loops.

Console-Output
```
Walk to stop Nr. 1. Talk about Language.
Walk to stop Nr. 2. Talk about FURTHER, FURTHER, FURTHER.
Walk to stop Nr. 3. Talk about Break System.
Walk to stop Nr. 4. Talk about Double Meaning.
Walk to stop Nr. 5. Talk about Embodiment.
Walk to stop Nr. 1. Talk about Language.
Walk to stop Nr. 2. Talk about FURTHER, FURTHER, FURTHER.
Walk to stop Nr. 3. Talk about Break System.
Walk to stop Nr. 4. Talk about Double Meaning.
Walk to stop Nr. 5. Talk about Embodiment.
Walk to stop Nr. 1. Talk about Language.
Walk to stop Nr. 2. Talk about FURTHER, FURTHER, FURTHER.
Walk to stop Nr. 3. Talk about Break System.
```

5.4. Theoretical Background

“Code is at the same time the description of an action and the action itself.” (Vee, 2017)

This performance was developed on the intersection of written code and the written event score of a performance. It relies on the core elements of logic that were explored theoretically: self-reference, loops, paradoxes and repetition.
The performance operates in a conceptually empty space and the fact that the gallery is empty refers to "simulation" through language: By talking to the audience, the script is “executed” on their minds like a poem and lets them imagine scenarios in the space.

Further, it is a comparison of creative writing and coding and a description of the point where they both break the system within they operate.

"A program can fail but a poem can not?" [46]

In literature, it is easy to break out of a logical system, like our perceivable reality, by introducing something that is not.

Example:
A blue elephant lives in the local botanical garden of Linz. Its ancestors were able to fly and inhabited Middle Europe when blue elephants were one of the most popular, but expensive modes of transport.

In creative writing, it is also possible to break the rules of grammar or the way words are spelled. In sum: if we first agree on a formal system within to operate, breaking out of it is the creative process.
But still, if seen from one level above, creative writing is exactly what it should be: it is done in order to break out of the logical system that it utilises. Language is made so we can describe the current temperature as well as the imaginary migration of blue elephants. That is the reason why words like "unimaginable" exist.

[46] Mignonneau, L. (personal communication, April 4, 2019)

And by that, breaking out of a formal system as a creative tool is swallowed by this very system and becomes part of it. Where is a true break?

Maybe we can find the true breaking point of a code exactly because it can fail.

A program, written in a coding language, for humans to read, for a compiler to execute. It can be creatively written and break the rules of the formal system (the programming language) but it becomes dysfunctional by the very act.
Creativity can be found in the writing style, in the clever, novel or even stupid way of problem solving. It can use the characteristics of language and functionality on itself (see 4.6.3 Quine).
But sometimes – deliberately or not – it breaks: an endless loop was created, there is a syntax error which causes the compiler to be unable to interpret the rest of the code. The physical limits of the machine on which the program is running is reached.

This is breaking out of a logical system: the code can not be executed. The simulation that was created within the programming language crashed. We stand in front of the ruins and can not observe any further. We find ourselves in the reality of a calculating machine that we didn't feed enough options to keep the simulation running.

5.4.1. The Text “A Repetitive Process” (see Appendix 3)

“A Repetitive Process” was written as the possible opening text of an exhibition on self-reference and on how extra value is created by the repetition of an action in the Gallery BB15 (Linz) .[47]
Besides repetitiveness, it revolves around the following two concepts:

Gödel’s Incompleteness Theorem:

“Can a copy and the original be the same thing?”

Code can be described as the logical basis of a simulation, the rule-book or the complete description of something that can then be imitated. The philosophical question addressed here is whether a copy can ever be the same as the original. Can an artificial, “coded” mind be the same as a natural one?

According to Gödel it is not possible, because nothing can contain the reason why it was made. A human mind is motivated by the question of its own origin, while an artificial mind would always know why it was created and who created it, and even if this information is not available to it, its origin would still be a different than the original mind. (Cryan, 2016)

We could say that an object can not contain its own “trigger”, the reason why it exists. A copy has always been intended as a copy, even if it perfectly resembles the original. The iterations in a repetitive process always differ, because they can be addressed distinctively (Round 1, Round 2,..) independent from what is happening within the process.

[47] Bevilacqua, D. Andreé-Pellerin, M. (personal communication, May 16, 2019)

Awareness as a Trigger for Action
"Does a tree make a sound when it falls and nobody is there to listen?"

In the works of Mischer'Traxler (see Artistic References) there is often an interesting loop that brings the final product back to its point of origin: The working conditions of the carpet makers are represented in the knots of the carpet, the outer layer of one chair gets cut off and becomes the inner layer of another. The baskets in "Collective Works (2011)" are only produced when somebody shows interest in them.

It is a philosophical question whether something only becomes what it is if somebody is present to observe it. Is art art if nobody is there to say it is? Does the context create the art? Does it have to be in a museum to be art?

In the case of a repetitive process, it becomes interesting to find out what triggers the process. What defines it and what ends it. The text explores the possibilities of processes that are triggered by awareness and ended when nobody is observing it.

5.5. Development

Bringing 3 Threads Together

Exercises of Self-Reference and Nothing
Since I started writing this thesis, I have been exchanging ideas with musician and performer Gabriella Gordillo. We share many ideas on process-based explorations. Among our favourite topics are "self-reference", "nothingness" and "abstraction/association".
We experimented on how additional value is created by using repetition in music, painting and performance. Also, we thought about the minimum conditions necessary to make something observable: What needs to exist in order to observe nothingness? From where does a reference start and to which part of itself does it refer to?

The Collective Manifestation of an Idea
In May 2019, I was invited to Gallery Kersnikova in Ljubljana. I met artist and curator Maja Burja who became an exchange partner. We discussed how things come into existence through the exchange of ideas: They "solidify" by repetition, communication and vivid imagination.
We also related this phenomenon to simulation, which creates possible worlds that only exist mentally, sometimes virtually, but reach a state of "realness" where they might be confused with reality.
We also talked about exhibitions that are art pieces by themselves and meaning that is created through referencing and comparison.

An Exhibition on Self-Reference
From February 2019, I had several meetings with BB15 members Davide Bevilacqua and Marie Andreé-Pellerin who were planning an exhibition on self-referentiality and additional value created by repetition.

My research on self-reference in code and poetry became the connector to this exhibition. First, we discussed my participation as an artist with one project, but I couldn't pin my research down to one piece. Instead, I wrote the text “A Repetitive Process” and we came up with the idea of a lecture performance that would happen before the exhibition as an experimental invitation-only format.

The Setting

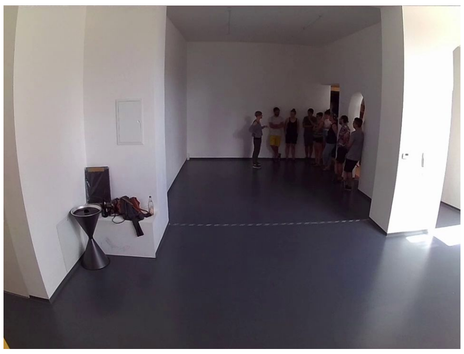
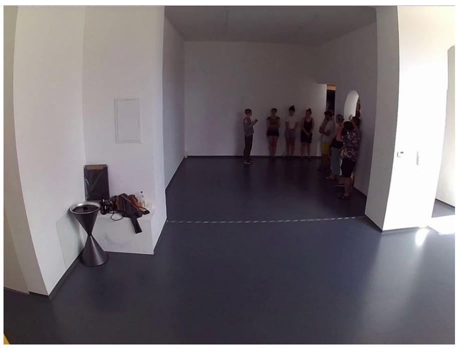
<u>The Format: Lecture-Performance</u>
I owe this project to my ten year experience as an art mediator. It was time to find out how a guided tour would feel like if there was no exhibition that provides content, but an interested audience. What happens to make a guided tour just about guiding tours?
The format of a guided tour seemed to be the perfect playground for the exercise I was going to perform. It would be a good reflection on my research and convey it to an interested group of people. No objects should be on display. The group should rather move forward in time than from object to object.

<u>The Frame: A Gallery-Space</u>
The gallery space sets the context for the experiment: This is a space where visitors encounter art. As a visitor, you invest your time and in exchange, the institution, an artist or a collective will offer any form of art and possible exchange with others.

Gallery visitors are trained to follow a certain behaviour of being open and respectful towards the art they are offered.

5.6. Documentation

Transcript

ROUND 1	*ROUND 2*	*ROUND 3*
Fig 9.1: Round 1/Stop 1 at "FURTHER, FURTHER; FURTHER", June 15 2019, Gallery BB15 Linz 03:34 Please stay on this side of the room. Can you all see me? Very good. Can you hear me? Also very good. Let's start. 17'' *03:51* *INTRO* *Self-Reference* I am doing my master thesis right now and there are a lot of repetitive processes involved. When you always refer to something over and over again, if you strip away all the unnecessary facts and get the essence in the end, or the function - and then you apply it again - it is a good way of showing how the thing itself works. If I say a **sentence** and there is nothing else in the conceptual space and I apply the sentence to	 *Fig 9.6: Round 2/Stop 1 at "FURTHER, FURTHER; FURTHER", June 15 2019, Gallery BB15 Linz* 17:01 Please stay on this side of the room. Okay, thanks for being here. 9'' 17:10 INTRO *Self-Reference* In my thesis and research, I was concerned about how to explain things and using the concept of them against themselves. So if I strip everything of its additional meanings, if I only use what is there, and I set it in an abstract space where there are no other connections, what is left? What does it do, when it refers to itself? For example a **sentence**. If you make a sentence about a sentence I think you learn a lot about what a sentence actually means.	 *Fig 9.11: Round 3/Stop 1 at "FURTHER, FURTHER; FURTHER", June 15 2019, Gallery BB15 Linz* *29:31* Thank you very much for being here! Please leave this place empty, 8'' *29:39* *INTRO* *Self-Reference* In my thesis, I was concerned with the topic of how to create meaning by stripping everything off everything that is not keeping its essence or its main functionality. And then I applied these to itself. Like a sentence - if you only have a sentence in your conceptual space, and then you use it on itself, on the **sentence** - it's a good way to explain what a sentence actually is.

itself, I learn more about what a sentence is. *Repetitive Processes, Code* Also my thesis is a lot about code. Code as a deterministic Languages, formal systems, Something where we make rules and then we try to run them, try to execute them. **A computer doesn't mind** if the same algorithm is executed over and over again. If it always leads to the same output, then this is the definition of an algorithm. But if a mind thinks about the same thing over and over again, does it also stay the same or does something change? 1' 5''	*Repetitive Processes, Code* And also I researched about repetitive processes. Processes that always refer to themselves. If you execute the same code on a machine, a **machine wouldn't mind**. Code is a formal system, something where you make rules and then you execute them one by one. And then it is deterministic, which means it will always lead to the same outcome. 1' 1'	*Repetitive Processes, Code* And a computer can be used to revise its own meaning. If you have a language for a computer - you give it rules that are executed on the machine and the always lead to the same outcome. It's an algorithm - something that always leads to the same output. **The computer doesn't mind** if something is repetitive - but humans do. Even though we always have the same input, something in the output will always change. There is something else that is coming out. 1'
04:56 *1. Language* The first thing I want to talk about here is language. I think nobody here is an **English** native speaker. So we all have at least two modes of constructing our world. Our own mother tongue and this one that we agreed on to use - for example now in this space. When I speak English there are many things changing: How a phrase, the words that I use - even my character changes. My Austrian character is different from my English one. And also the atmosphere changes. So maybe you know that as well. So, we agreed on this language for our communication. The interesting thing about languages is that I say something and it triggers some kind of	18:11 1. Language With my language, I can create a **hallucination** in your brain to start something that happens in your brain. you also can create something as a reaction. So this is the first thing I want to talk about, language. I think there is no native speaker of **English** here, so we all have at least 2 ways of how we perceive the world. We have our own mother tongue and we have this common language that we agreed on, that we use to create this simulation. So I can invoke some different concepts with the language, that's how we communicate.	30:39 *1. Language* The first thing I want to talk about is language. So we are speaking **English**, although nobody is a native speaker - so we all have at least two ways how we perceive reality. If I talk in English, I see that I use a different voice, even a different character than my Austrian character. The words are different, the sentence structure is different, the atmosphere changes. With this possibility of language, I can put a **hallucination** in your brain!

hallucination in your brain - your start to have an idea by that. So it's kind of running a simulation.		
Now that we are in this space - let's make a thought experiment. Let's imagine there is an artwork here. By now I just said artwork, so it's an empty container. I can say there are a **pedestal** and a sculpture on it. The sculpture is very big - it **goes until the ceiling**. It is from its volume like a sphere and the material is like the pedestal - **metal**. And there are four faces, on each side one. So they face in four different directions. And they are all kind of emotionless. It's not the same face, all four are different. **I don't have to explain how the faces exactly look**, because I think we all have a big repertoire of how faces look like in our brains. So you can put your own faces there.	For example, this place is empty, it looks empty. But let's say we want to imagine an artwork in here. We could start with a **pedestal**. And we start with something very big, it has a volume of a circle and **goes until the ceiling**. It's all from **metal**, even the pedestal. And it has four faces on all sides. And these four faces. I think **I don't have to describe what faces look like** because we have a quite big space allocated in our brain to imagine and store faces. Just put your own faces that you have in mind there. They are all different.	Let's try this as a thought experiment. Let's create a sculpture in here. Let's say there is a big **pedestal** and it has a sculpture on top. It has the volume of a circle and it **goes until the ceiling** - and it's from **metal** and on all sides of the circle, there are four big faces. **I don't have to say how the faces look like** because our brains have big repertoires of faces, so you can all imagine your own faces and put them there.
Do you all have it? Do you see it? OK. **For now, we keep it here.** It's here in this space now. So, that means, when we continue our tour, we move on the side - and leave it here. [Group moves] I hope you can also see me on the other side! 2' 4''	Do you have it? So let's start. **We keep this here for now.** So let's say the place is not empty anymore, we leave it here. And when we move on then please also take care of it. Can you see me over there? Okay. Take care! [Group moves] Davide: No flash, please! 1'35''	**And then we keep it here for now.** And we continue, we move on. For now please keep the sculpture in mind. [Group moves] 1'21''

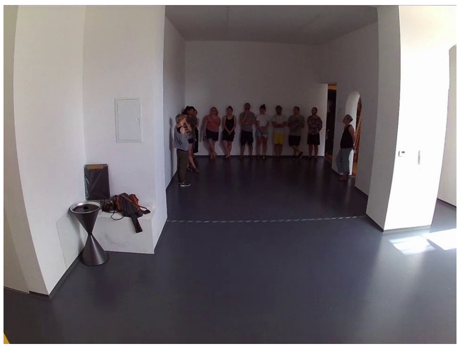
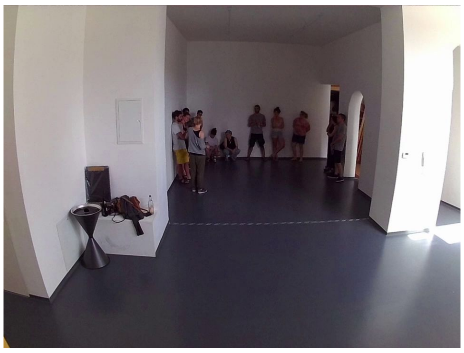
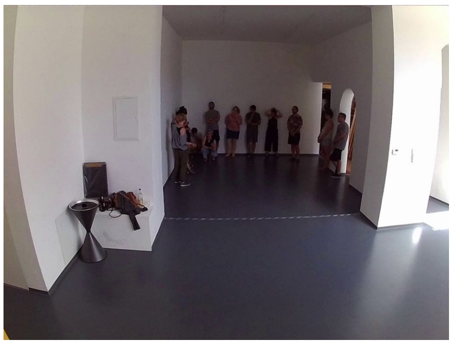
		

Fig 9.2: Round 1/Stop 2 at "FURTHER, FURTHER; FURTHER", June 15 2019, Gallery BB15 Linz	*Fig 9.7: Round 2/Stop 2 at "FURTHER, FURTHER; FURTHER", June 15 2019, Gallery BB15 Linz*	*Fig 9.12: Round 3/Stop 2 at "FURTHER, FURTHER; FURTHER", June 15 2019, Gallery BB15 Linz*
07:00 *2. FURTHER, FURTHER, FURTHER* **You all got my invitation with the title FURTHER, FURTHER, FURTHER.** So where do get to when you go step by step. When you research, when you read - where does it bring you? Does it bring you anywhere? And where are you when you are coming back to the starting point? When you start again? Did everything vanish what you researched before or is it still there? Or what's the meaning that is left?	*19:46* *2. FURTHER, FURTHER, FURTHER* **You all got my invitation to the topic or the name of this event "FURTHER, FURTHER, FURTHER".** That was also a big part of my own research. If I go further, I learn more, I get new knowledge. But what happens if I go back to the beginning? Is there still some meaning left? Or is it all gone? What is left?	32:00 *2. FURTHER, FURTHER, FURTHER* **You all got my invitation here with the title FURTHER, FURTHER, FURTHER - that's the name of this event.** So in my research, I try to always get further, to get more knowledge to learn something. But what would happen if I direct myself back to the starting point? Would all meaning be lost? What will stay behind?
Let's also talk about **context** and rules. Context is, for example, this space. It's a **gallery space**. We know how to behave in a gallery space, or that we can expect art to happen here. Or that you meet others who are interested. So all these rules, we know them. And then we can apply them, we can change them.	That also makes me think about context. The **context** is that we are here in a **gallery space** - that houses art maybe. There are the rules of a gallery space. How do we behave in it? What do we expect from it? Maybe we expect other people here who we know.	And then that also makes me talk about **context**. We are in a **gallery space**. The gallery is there to house art and there are rules that apply in the gallery space: How we behave, what we expect from it. Maybe there are people we meet. And we learn these rules.
It's the same with language, you learn the **grammar** for example, you learn how to write and then you can use it.	This context brings some rules. And rules are applied to many different things - to language for example. We learn the **grammar**,	This applies to different systems, different context: For example for writing. I learn the **grammar**, I learn the rules of the language

For me, this knowledge is a good starting point to write a **poem**. When I know the rules of a language, I can know how to make functional sentences, but I can also know how to break the rules, how to create something out of them. Same also with code: Again, the code is also a language that makes everything very formal and well structured, always leading to the same result. So, it makes it also possible to play with logic, to explore paradoxes for example. To use this logic and then apply this logic against itself. For example, I can say "I am lying" and it is not true - or what is it? You can play and create new things out of language. That's what writers and coders might have in common - they make use of the structure they are working with and then applying them and breaking out of them sometimes. For now, we still have this **sculpture** here in the room. But let's forget it. Let's say we put it out: There are 10 people coming in, dismantle it and move it away to another place - now it's gone. It was super fast! Please come with me to the other side. [Group moves] 2' 21''	we learn the rules and then we can use them. Or in code, we can learn the rules to run the code. And the creative part, for me how a creative **poem** works is if you use these rules and break them or use them against themselves. That's also what a writer is doing - but also a coder. Mastering the rules and creatively breaking them. Also, the same thing happens when we go to a gallery space. How is the art created in here right now? But for now, we still have this **sculpture** that we created in here. Let's put it away. Let's say it's a quite fast process, let's imagine there are 10 people who came to dismantle it and now they put it back into the care and bring it to another exhibition. So this space is empty now. It's even emptier than before. Because we extra emptied it. [Group moves] It's very good because I can see you here instead of the sculpture. 1'56''	and then I can use them. I can play with them. If I code I can play with the logic that I learn - how it is working. And that is for me a way how to create a **poem** for example. To say: I play with the rules, I change them and this is the creative process in it. So an artist or a writer can be somebody who knows the rules, using them and also sometimes using them against themselves. So, the **sculpture** is still here for now, but we want to get rid of it. Let's say it is gone. It was a very fast process. There were like ten people coming in and they put it all away. They put it in the car now it's leaving to the next place. It means that we can now move through the space. [Group moves] It is nice that I have you all here now in the space where the sculpture was! 1'46

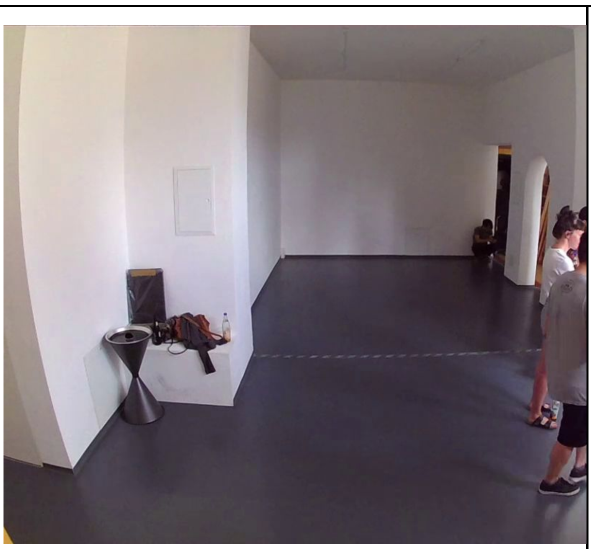
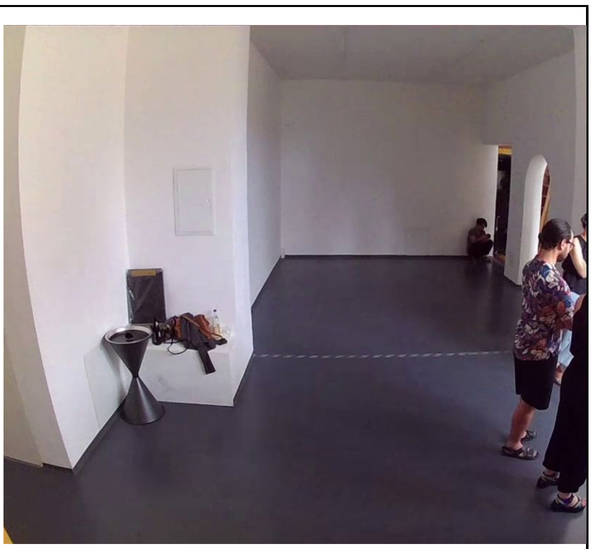
Fig 9.3: Round 1/Stop 3 at "FURTHER, FURTHER; FURTHER", June 15 2019, Gallery BB15 Linz 09:21 *3. Breaking out of a System* Here I want to think about **how to break out of a formal system**. So there are a lot of rules that we make in order to simulate something, in order to understand something.	*Fig 9.8: Round 2/Stop 3 at "FURTHER, FURTHER; FURTHER", June 15 2019, Gallery BB15 Linz* *21:42* *3. Breaking out of a System* So, the next thing I want to talk about is **how to break out of a system**. So we can create simulations, we can observe them and we can use them to see something that physics doesn't allow. We can use simulations to try out different versions of reality.	*Fig 9.13: Round 3/Stop 3 at "FURTHER, FURTHER; FURTHER", June 15 2019, Gallery BB15 Linz* 33:46 3. Breaking out of a System Let's talk about **how to break out of a system**. And thank you very much!

09:21 *3. Breaking out of a System* Here I want to think about **how to break out of a formal system**. So there are a lot of rules that we make in order to simulate something, in order to understand something. I think computation became very sexy in the 20th century because it was useful for constructing many different realities and thinking about if that might be true, or that, or that. I don't have to write down and reason all the possibilities by hand, instead, I can run it and I can observe it. If we think like this - what we have here is also a simulation of reality. It's built in our	*21:42* *3. Breaking out of a System* So, the next thing I want to talk about is **how to break out of a system**. So we can create simulations, we can observe them and we can use them to see something that physics doesn't allow. We can use simulations to try out different versions of reality. I think computation got very sexy in the 20th century because the rules were not only formalized, they were also applied. You can just watch while they are executed. It also helps us to perceive things that we in our reality are not able to perceive - like infinity. Let's say we also create a simulation here - somewhere in the **neocortex**. And we look

neocortices and then we look out of our **eyes** and then see the world. It's a bit hard to break out of that.	out of our **eyes** and perceive the world, we construct it. But how can we go out of this?
Let's say we are in a computer simulation and we want to - for example - observe the concept of **infinity**. A very easy way to do it is to code a loop that does not have a breaking condition. We could write "execute this over and over again until that one changes." But that one never changes, so - it repeats until the system breaks. The program is broke, the computer shuts itself down and we are suddenly outside of it and question - "What happened?". I can not observe it anymore. I just broke this simulated reality.	We can try to simulate this scenario and then see. For example, if I write code I can easily create an **infinity** by creating something that doesn't have a reason to stop. For example, I say: "repeat this over and over again until something here changes." But if that here never changes, then it would just fill the computer until it breaks." And then you are suddenly confronted with this break, something that you can not do in the reality we live in.
That works with loops, that works with **recursions** - a recursion is a function that always calls itself - where you go deeper and deeper into itself.	The same goes for recursions for example. A **recursion** is a function that always calls itself. And if there is no breaking condition, it can call itself over and over again until it also fills the memory space and then we also break out.
But the computer doesn't mind if the code is running again and again - but if we would do something again and again and again, we wouldn't be able to continue indefinitely. We would die at some point, or just get bored and have a **coffee**.	So this code, when it's executed, it will always do the same. It doesn't mind if it's running again and again. But we humans we can not do this. If we would infinitely repeat an action, we would get tired or die or get a **coffee**.
That brings me to create simulations of our own **mind**. How to do that? Let's say we use computation as a tool to find out what a mind is. We create this code that can simulate reality as close as possible. And in the end, we will find out that there is something missing, <u>the purpose of why it was created.</u> We don't know the reason why we were created, the simulation always knows.	The machine would nevertheless do that. So it's also used as a means to create something like a mind to see how a **mind** works: Let's put everything in a code - everything that we think contributes to how our minds work - an artificial mind. Could that artificial mind be the same as ours? One thing that will be inherently different, <u>that it will know it was made for a certain reason.</u> We don't really know what the reason for our existence is. The computer knows that it was created to recreate us.

And all these attempts of creating perfect simulations, they just help us understand **what we are not**. That there is something missing. So it's just another thing to exclude. How to break out of these systems? Let's continue. [Group moves] 3' 23''	So we can create more and more instances, examples of minds and then observe them and always learn **what we are not -** or what is the difference between us and the artificial mind, or this uncanny factor. At least we know that this is not what we are. So we can cross another thing from the list that we are not. Please follow me. [Group moves] 3'11''
 Fig 9.4: Round 1/Stop 4 at "FURTHER, FURTHER; FURTHER", June 15 2019, Gallery BB15 Linz *12:44* *4. Double Meanings* This is about double meanings. We are next to the **bar**. Double meanings in the following sense: When I was writing my thesis and doing my research I was wondering if I am writing as an **artist or an observer**. Am I the one who is criticizing and writing theoretically or am I the one who is doing art? There is a quote by Barnet Newman: "Aesthetics is to the artist what ornithology is to the birds."	 *Fig 9.9: Round 2/Stop 4 at "FURTHER, FURTHER; FURTHER", June 15 2019, Gallery BB15 Linz* *24:53* *4. Double Meanings* Now we are here, next to the **bar**. Double meanings in my research meant, that I was not sure if I am the **observer or the artist**. Because I started with the process of writing and examining something and then I wanted to become the person who creates the art. There is this quote by Barnet Newman "Aesthetics is to the artist what ornithology is to the birds". It means the artist should not be occupied with aesthetics but just do.

If you are doing the one you should not do the other, or: the artist should be free of concerning themselves with aesthetics. So what I did was defining my research topic and then try to learn as much as I can about it, driven by curiosity. Reading the books, watching the movies, getting into the science, writing my own texts, seeing fiction, science fiction, fan fiction and then I decided to forget everything. I rest myself and tried to **become my creative process, become the practice**. **Like not being the container anymore, but the object that is contained.** **Not being an array holding the number one, but being one.** **Not becoming the describing factor but the thing that is described.** And here, everything that feels right is what you should do. There are no questions anymore. Just forget what you learned. That's when the creative practice **becomes alive**, let's say. And then there is again the moment where you can **lean bac**k and **observe**. Now you can see things from two different perspectives. [Group moves] 2’8’’	So how can I do that? This is the process that I came up with: I researched, I read everything there is in the field, I watch the movies, I look at the artworks, I see the science, the science fiction, the fan fiction and in the end I try to forget everything. As soon as I forgot everything, I am able to **become my own practice**. I can start to do. **I can be the object inside, and not the container anymore.** **If there is an array that carries the number one, I can become one. Instead of the describing factor, I can be the thing that is described.** As soon as that happens, everything is okay. I forgot about all the artworks that were there before and just make whatever feels right. And in the end, when my practice **became alive**, I can **lean back** look at it. That's when I become the **observer** again. Now we move to the last space. [Group moves] 1’42’’

 *Fig 9.5: Round 1/Stop 5 at "FURTHER, FURTHER; FURTHER", June 15 2019, Gallery BB15 Linz* 14:52 *5. Embodiment* **Thank you very much that you not only lend me your minds here but also your physical presence** on this very warm day. You are here now, you saw the rules of the game. How are you right now? Where do you stand? What is the last thing you thought about before you were here? Is there anybody who wants to say something? *Gabi*: I thought we are so close and we didn't have to walk. [laughter] *Marie*: If we were in the first place, would it be the same? And how is it to walk in this quite small space? *Loren*: It's pretty funny because it's empty now. And the next point is very close. *Davide*: But it was not empty. The sculpture is there!	 *Fig 9.10: Round 2/Stop 5 at "FURTHER, FURTHER; FURTHER", June 15 2019, Gallery BB15 Linz* 26:35 *5. Embodiment* **Thank you all for not only lending me your mind but also your physical presence** here. So now that we already know the rules of the game, how do you feel? Where are you standing? What was the last thing you thought about before you arrived here? *Gabi*: I think you forgot to say what a beautiful day! [laughter] *Julia*: And I thought you were putting us in a loop. *Loren*: It was not actually a loop because when you explain in a different way, we get the same information another story - you tell it differently. *Julia*: And we are standing in different spots. And I was listening to different parts of your explanation. *Davide*: I didn't remember that the big sculpture was metal in the first round. And

Loren: It was full of us! *Julia*: The guided tour needs a guide and a group and a space. *Hess*: At some point I forgot about the actual space. Are you back now? *Hess*: Yes, now that you asked it? But this is no all, pleas follow me. [Group moves] 2’ 9’’	in the second it was suddenly metal and it changed a bit. And then I am asking myself: will there be a round sculpture where I got everything? Or will there be always a process of change? *Julia*: We will probably find out now. *Maria*: We are looping, but things are also a bit changing, we are closing the loop. How do you feel about walking? *Davide*: Well, I wish this place was bigger. We'd have more distances. Do you remember where you were standing last round? Can you stand there again? [People change places] Thank you very much, but this is not all, let's continue. [Group moves] 2’56’’

5.7. Analysis

The performance initiated an interesting discussion about machine mimicry, the collective construction of art, space exploration, processes of learning, effects of repetition and self-referentiality.

After the event, which took 35 minutes, all 13 participants stayed for another 50 minutes to participate in the feedback round.

5.7.1. Mimicry

The first perspective on the event is to see it as a theatrical exercise of acting like a machine. The code which is executed by the group is the script. Written algorithms have certain things in common with classic theatre plays. They are both one-way instruction based, as the programmer and screenwriter instruct the performer. Also, in both cases the script is written first and then put into action. The originator of the text then observes, and changes the script if the performance needs adjustment.

In this case, the screenwriter is an actor who performs the role of the machine for which the script was written. The machine/actor delivers the desired output from the program which is to walk and talk according to a predefined pattern presenting a predefined content.

The attempt of conducting a performance and executing a script like code on the CPU of a computer restricts the performer from social interaction, soft skills like interpretation of the audience's behaviour, improvisation or reaction to unexpected inputs from their environment. The mimicry is indirectly expected from the audience (they mimic the performer) and thereby, the same behaviour is expected from them.

In reality, the strictly deterministic performative style was a good goal to work towards, but both performer and audience lacked training to conduct it that way. The performer did not stick to the script word by word and tweaked it every round. Thus, the audience was confronted with the rule of machine mimicry while the performance started, so for them it was a process of learning.

The group developed two contradictory behavioural patterns:
A: "I know we are trying to mimic a mechanical repetitive process. I go along with it and pretend I am a machine."
B: "I know we are trying to mimic a mechanical repetitive process. I also know that a group of humans is incapable of doing so. I pay attention to anything that shows exactly that: errors, differences and new information."

If we argue that behaviour A would be more beneficial for the experiment: The repetitions progress without interruption. Variations are accepted by the group as a negligible human error. It is the "polite way" of participation – let the performer trick us.

Unfortunately, behaviour A doesn't help the experiment much. It would create a group of people mimicking machines in a quite unchallenging manner. Walking and listening don't require extensive use of one's brain.

Amir Bastan mentioned afterwards that they liked being tricked very much (behaviour A), but got constantly carried away by comparing what was happening to their memory of the preceding round (behaviour B).
The guide's goal was also to create a tour experience of good quality, keep the audience engaged and present the content in an interesting manner. This goal is contradictory to the aforementioned definition of machine imitation by the performer.
How to solve the contradiction? Enacting an entertaining robot was clearly not the aim of the exercise.

The performer in this double role, and as a human outside of any role, cannot not feel their audience. That's why it was a challenge to provide the exact same information in the second round. It was clear that the group had heard it before. This lead to each round becoming a little bit shorter. Errors like "Here we are again.." were made often in the little sentences between information presentation.

Marie Andreé-Pellerin suggested that playing with the audience's attention would have been a useful tool. Nobody paid perfect attention all the time, but at the same time, most of the

information was remembered, especially the little errors. While they liked to point them out (behaviour B) they enjoyed being tricked (behaviour A).
Afterwards, a discussion took place on whether a robotic voice that is set on loop-mode could be a proper substitute, but the participants stated they felt more comfortable human guide.

Two participants (Gordillo, Bergantini) had a discussion about the role enacted by the performer. Gabriella Gordillo argued that it was the role of a machine, because of the repetition without acknowledging that the group was already well informed about the content. Loren Bergantini said that it must have been the human role because repetition, especially as a practice of presentation, is a very human act and the audience was part of the “rehearsing process”. Also, it’s a guide’s daily job to repeat the same information over and over again, but normally, the audience is unaware of this, as they take the tour only once.

5.7.2. Repetitions

Endless loops can be best understood by executing them, not by reading.

The event was based on repetitive processes, but only had the smallest amount of repetition: two to three times per stop. It could have been five or more. The opinions on how long it should have taken were different, also depending on the goal of the exercise for both performer and audience.

Possibility: Endless Repetitions
The tour could have been repeated until it naturally breaks due to fatigue, the disinterest of all participants or the guide. Trying to avoid “breaking out of the system” would become an endurance performance.

Possibility: more than three repetitions
Gabriella Gordillo argued that all three rounds fulfilled different purposes, despite their content being the same:

> “In the first round we observed, then we learned the rules. The third round wouldn’t be boring at all, because that’s when the audience becomes free - Free to play with the rules.” (Gordillo, 2019)

What is the minimum requirement to get a feeling for eternity? - repeat something two times and don’t tell your audience how many repetitions will follow.
The intention was to make the concept of repetition clear and to let the participants learn the rules. By not informing them about the number of repetitions, infinite repetitions became an optionThis way, the concept was included without the need of explaining and of course without fully executing it.

5.7.3. Self-Reference and the Effect of Repetition

Another perspective is the tour as an experiment of repetition and self-reference.
This event was the result of research on rule systems and conventions. The goal was to see and think about structure only and to avoid anything that would only fill and weigh on the structure – anything that could be considered “content”.

In order to avoid unnecessary references, all the topics mentioned within the rule system were about the rule system. So, the container became the contained object.

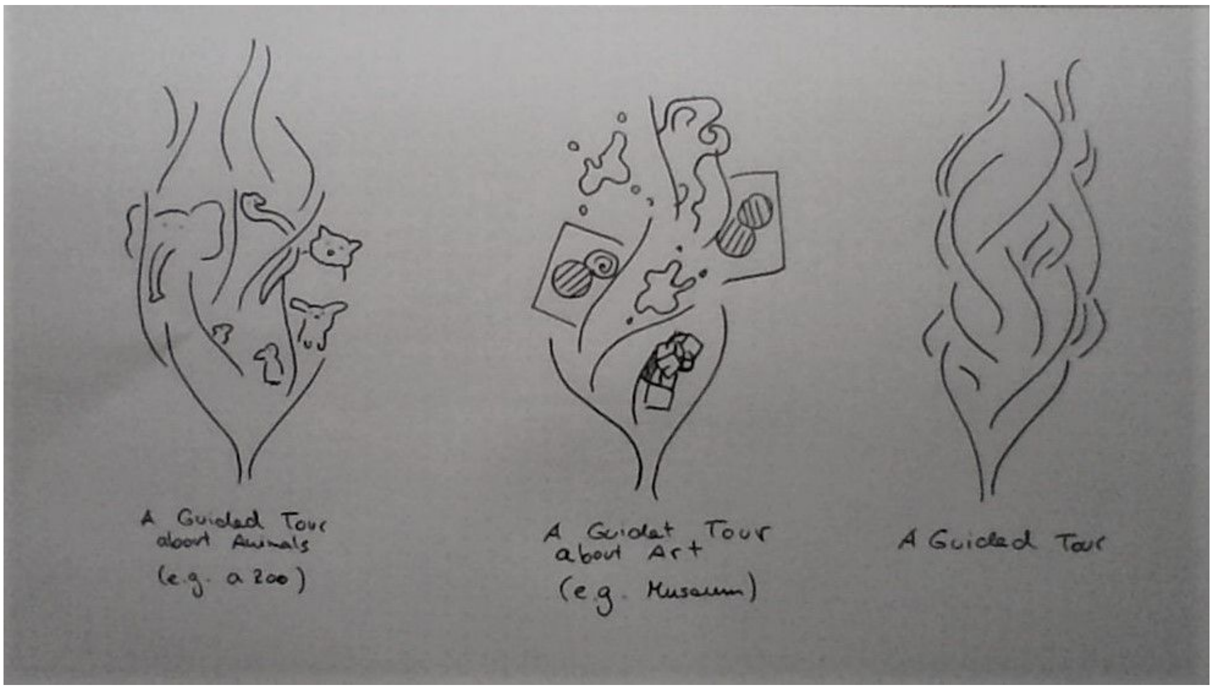

A Guided Tour about Animals (e.g. a zoo)
A Guidet Tour about Art (e.g. Museum)
A Guided Tour

Fig 10: Illustration of semiotics of a guided tour by the author

The venue (a gallery), the *group* (people interested in conceptual art), the *act* (guiding, walking stopping), the *time frame* (one hour on a Saturday afternoon), the *guide’s knowledge* (research and preparation), the *language* to convey information (English), the *intention of the tour* (self-reference, repetition and allocation of information in space), the *beginning and ending* (how to start, how to end), the *role of the guide* (artist or theoretician), the *role of the audience* (recipient of the tour) – all became the content of the tour.

So theoretically speaking, the format of the tour was the content of the tour. But how would this conceptual self-reference manifest itself in reality? What would draw the attention of the group and the tour guide?

Where does self-reference stop and the tour begin? Amir Bastan pointed out that in his opinion, the “perfect loop” is a still image. The still image was meant to be a philosophical figure, but exactly touched the point why the tour needed to include content at all: first and foremost because otherwise, it wouldn’t have an element of time in it and then - cease to be a guided tour - which is by definition a time based format.

How self-referential can speech and its content be?

X	X	X
absence of content = silence	generic/placeholder content	personal examples

Rule 1: The more colorful the examples are, the more self-referential they should be.
Rule 2: No content might mean anything and is therefore of little relevance.
Rule 3: The content of the speech should tend to the middle: generic, placeholder content.

Self-reference comes into play again, because the research for the thesis became its practice.

5.7.4. Information

While the instructions are clear and always the same, move - talk (one place, one topic), the information is the same, but the words used to convey the message are not.

Example:

ROUND 1:	It's the same with language, <u>you learn the grammar…</u>
ROUND 2:	… to language for example. <u>We learn the grammar</u>, …
ROUND 3:	<u>I learn the grammar</u>, I learn the rules of the language ...

In this example, three different grammatical persons are used rhetorically to exemplify the possibility of something.

It is interesting to discuss whether or not sticking to a script still fits the concept of "executing a program". Should the performer rehearse until they are capable of performing the script perfectly and indistinguishably from each round before?

One could argue that the walk symbolises a mechanically reproducible, deterministic act and thereby, every round has to be the same as the one before. This argument is a strictly conceptual one.
It bears in mind the model for the whole performance which is the generic machine that repeats an act automatically without ever learning from it, unless it is instructed to do so.
For a human, doing exactly that is impossible: Being presented with the same input over and over again, recognition and/or muscle memory happen automatically.

But in this performance, there are no machines involved. It is one human repeating information to a group of other humans.
The performer mimics the computer and its one-way communication (delivering the desired output) by ignoring the fact that her audience is capable of remembering, learning and responding.

The tour was criticized by Marie Andreé-Pellerin for not sticking to the principle of self-reference completely. For them, the comparisons between computers and humans were

too far-fetched and did not help the self-referential exercise. It is a valuable comment when it comes to reduction and “decluttering” to make self-reference more observable.
Still, the reference to computers is legitimate, because the concept of the tour relied on the idea of the lowest common denominator of applied formal systems and human reasoning, which is logic’s purest form.

So how relevant was the information given in the tour?
As discussed before, the intention was to construct and observe a conceptual space and its changes when we engage with it. The information was there to keep it standing and observable and fulfilled a secondary, distractive role.

Gabriella Gordillo emphasized the role of information in order to understand the structure:

> “By your presentation you gave us the tools that were needed to interpret the action. The information was very relevant. If the tour was about something else, we could not make the self-referential exercise. You talk about what you are doing!”
> (Gordilllo, 2019)

5.7.5. Narrative by Repetition

The information that was presented in each round did not change, but it was repeated. However, through this repetition, the context changed. A narrative by repetition was established.
The information given in each round was very dense, so it took several iterations to understand? all of it. Some information could only be understood in a later round, because the context had changed.

The “artist/presenter dichotomy” in stop four is a very good example: Both times, it was about how a person is a researcher on repetition and self-reference and an artist who enacts them. In the first round, the group and the guide identified themselves more with the role of a researcher. The second round was an actual loop: the act of repetition and self-reference made the guide an artist.

The same goes for the introduction: The topics of repetition and self-reference were introduced as theoretical concepts. In the second, but the latest in the third round, this introduction became the starting point of a loop. The group was practicing loops and self-reference instead of just passively listening.

At the second stop, the title FURTHER, FURTHER, FURTHER was explained and might have been understood as a linear function, a straight path into nothingness. Later, the same content gave the impression that the path represented a circle and ends with its own beginning, making the one who walks it one round richer.

5.7.6. Collective Construction of Art

One part of each round in the tour was a language experiment that turned into an exercise of mentally constructing art, similar to the attempt of Maja Burja and Urška Aplinc in their performance “Variable”.

An empty gallery space filled by a group of people could be the basis for art to emerge through communication by imagination and agreements. In FURTHER, FURTHER, FURTHER, the tour guide created a story and thereby, an artwork for the audience.
It could be explained as the job of an art mediator: By adding context to an art object through stories and explanations, the audience is able to grasp it. The only difference in FURTHER, FURTHER, FURTHER was that there was no reference object. The starting point of the art piece was a mental one and stayed in mental space.

Example
ROUND 1
Let's imagine an artwork here. I can say there are a **pedestal** and a sculpture on it. The sculpture is very big - it **nearly reaches the ceiling**. It’s volume is that of a sphere and the material is like the pedestal - **metal**. And there are four faces, on each side one. It's not the same face, all four are different. **I don't have to explain how the faces exactly look**, because we all have a big repertoire of how faces. So you can put your own faces there.

ROUND 2
But let's say we want to imagine an artwork in here. We could start with a **pedestal**. And we start with something very big, it has a volume of a circle and **nearly reaches the ceiling**. It's all from **metal**, even the pedestal. And it has four faces on all sides. I think **I don't have to describe what faces look like** because we have a big space allocated in our brains to imagine and store faces. Just put the faces that you have in mind there.

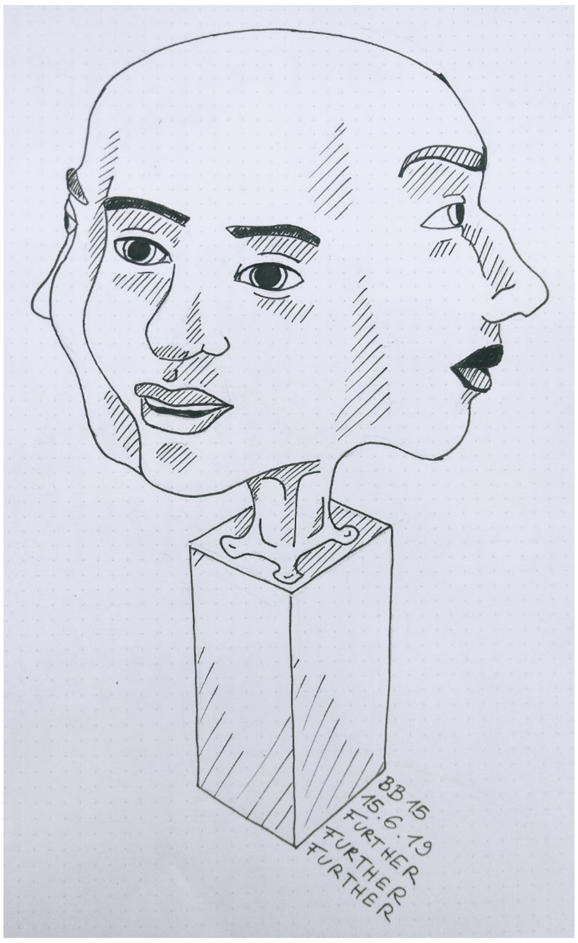


Fig 11: Illustration of the imaginary artwork described during the event "FURTHER, FURTHER, FURTHER" on June 15 2019, at Gallery BB15 in Linz

Round 3
Let's create a sculpture in here. Let's say there is a big **pedestal** and it has a sculpture on top. It has the volume of a sphere and it **nearly reaches the ceiling** - and it's from **metal** and on all sides of the circle, there are four big faces. **I don't have to say how the faces look like** because our brains have big repertoires of faces, so you can all imagine your own faces and put them there.

"*Why not something more generic like a square?*"
- asked by event participant Fabricio Lamoncha

It is arguable that the shape of a generic artwork is a square. The general idea of an artwork can not have a shape. But this argument leads in an interesting direction.
The purpose of the collectively imagined artwork was not that of a placeholder, but to evoke an image in the participants' minds that none of them had had before.

It was meant to prove how language can evoke a thought that we could all agree upon and reference to later. It symbolized the creation of an idea out of nothing. For this purpose, it was sufficient.

Unfortunately, for the concept of the tour, it was rather counterproductive. While the major part of the transmitted information to the audience was generic, this sculpture stood out as a clearly defined example.

In the feedback round, the sculpture was mentioned positively, because it worked well as a thought experiment. Hess Jeon mentioned that it was the thing they remembered best and that they wished that the tour would have included more such examples.

It's an interesting observation, because it contrasts the desired behavior of the audience with their actual behaviour. The desired behaviour was to make them stick to the exercise of repetition as close as possible and the actual one was the pleasure they got from sensorial stimuli and the collective experience of creating this sculpture.
Davide Bevilacqua and Amir Bastan experimented by themselves by checking whether the sculpture would change in their minds by listening to the description of it given in ROUND 1, 2 and 3.

5.7.7. Space exploration: How to locate information in space

"So was this tour a space exploration or an exercise of repetition?" asked Gabriela Gordillo.
The exercise of repetition was a set goal. The space exploration was a natural side effect of the format. A guided tour can be considered a space exploration. Much more so if there is no distraction from the space where it takes place. (A distraction would be any content attached to the tour, like art, artifacts, sights,...)
Also, the smaller the space, the more thoroughly it can be explored within a given timeframe.

One of the questions by Fabricio Lamoncha afterwards was: “Does it have to take place in a gallery space?”

Yes, because the space is an essential part of the self-referentiality of the guided tour. It sets the necessary conceptual brackets. Our behaviours and expectations towards the tour are shaped by the rules of a gallery space. Exactly because there is no art where it’s expected to be, it is easier to imagine it, like a blank paper and a pen next to it.

An empty gallery space deprives the senses of distraction and directs the attention to the performance. If any desire for sensory stimulation is emitted to the surroundings, the senses can not easily hold onto something, but will be reflected back to the exercise.

“Does it matter if there is any art in here or not?”
- asked by event participant Davide Bevilacqua

This question was asked in the context of the sculpture that was collectively and mentally created at the first stop in each round. As we proved that we can imagine an artwork, why can’t we pretend that existing art is not there?
Again, the answer would be that having a space clear from direct references is preferable, as in the end, it leaves more room for imagined references.

A guided tour locates information in space. It constructs the space in which it takes place. Before, the area can be anything. Afterwards, especially the places that were discussed during the tour have another meaning for the audience. Often, artworks or artefacts change their meaning, as they are brought to life by the guide through stories and context.
In this self-referential tour, each stop was another step towards understanding the act.

The event also was an exercise of sculpting and shaping the space by exploring it, locating non-existing imaginary objects in it, emptying it again, going back to the same spot with different thoughts, trying not to get distracted by it, being contained by it, hearing one’s voice echoed in the empty room, sensing that the space is too small for walking and halting five times per round.

Example:
ROUND 2, Stop 2
“Removing” the mental sculpture that was created in the stop before:
So this space is empty now. It's even emptier than before. Because we extra emptied it.

5.7.8. Learning

Leaving all the disguise as machine mimicry, space exploration and the forced collective creation of art aside, the event was about learning.

Repetition, exercise, taking part in a tour, rehearsing, practice, communication ideas, research, the cycle of observation and action, transmitting information, the audience who gets to know the rules of the performance – all of it is about learning.

We can pretend to be machines, but we are not, because we involuntarily learn. In a wake state, our senses need stimuli. Machines don’t learn unless they are clearly instructed to do so and don’t mind having no stimuli.
The logic performed by the machine is something that humans discovered, programmed into them and watch them execute. The machine that is programmed to perform the logic learns nothing by doing so, but the observing human does.

According to Loren Bergantini, the tour was an exercise of learning for the presenter, as she could go through the contents of her thesis and put some parts of this research into action and thereby, understand more about it.
It was also an exercise of learning for the participants: Through repetition, they learned the rules and were always able to change their focus on a new element which was of their interest and not yet understood.

The tour also proved that learning by association works very well. By allocating information in space and by creating images that help to store the new information, the audience is able to better remember what happened during the tour.

Possibly, if the whole event had been completely self-referential without personal experiences or links to anything outside, it would have been hard to remember what happened. Participants could only say: “I took part in an exercise of self-reference.”

It was an exercise of machine mimicry, but there were unexpected topics that I want to explore further now: how to shape a space by exploring it and assigning it meaning as well as the collective construction of art.

6. Personal Project: For while { Exception println #update let !doctype @type } ;

Poetry Evening at Art University Linz, May 16, 2019

Fig 12: Participants at the poetry evening at Art University Linz, May 16, 2019

Code Poetry for me was the step into an analysis of language. I was interested in the creative use of it as well as its function in logic as a form of agreement of what we know. I saw the use of language in poetry at one end of the spectrum, while code as a way to instruct a machine on the other.
Both can be tools to create simulations: poetry inspires to create imaginary worlds that nurture creative processes. Coded simulations only run on a computer if they are error-free, so they can help us to understand the laws of the world we live in.

A code poem is the combination of what is expected from executable code on the one hand and abstract concepts formalised in language on the other. Its medium is written language that uses the syntax of programming languages for linguistic experiments, as well as aesthetic ones.
I was never interested in code poems per se, more in the discussions that they open. Nevertheless I accumulated a collection of code poems - from publications, by writing them myself and by requesting them from friends.
For me, these poems symbolically stand for the variety of possibilities that language offers. Here are some of them that I was talking about throughout this thesis.

- Through language, a simulation can be run on the listeners/receivers mind.
- Anybody who wants to participate in an interaction based on a certain language, needs to know its rules. These rules can be used but also subverted or neglected in a poem.
- Through language, we can transfer information and keep knowledge.
- Through language we can express ourselves - on a level of facts and feelings.
- Language constructs reality, each person uses language differently, thereby constructs their own reality.
- Language is comprised of symbols and their meanings.

- The attempt to construct a language that is universal and objective so far always failed.
- The attempt to get different interpretations of a sentence by different people always succeeds.

So the wish to learn about language and its impact on art and logic lead to naming poetry and code as two juxtaposing fields of it. Next I started looking for moments where they came together. A code poem is the most direct combination of the too.

6.1. Project Outline

During the research on code and poetry, one of the questions was, how poems or short stories would look like if they were written in the style of programming languages.
Me and Fabian Frei, another student at the Interface Cultures department who was researching poetry in the context of machine learning, organized an event for creative writing. We invited art and computer science students, writers and friends to participate and write their own poems. Then we asked them to read their poems before collecting them anonymously.

I prepared 14 algorithms printed on paper, that would work if run by the right compiler to give the participants a reference on how executable source code looks like. The codes were written in java, html, glsl, c++ haskell and Processing. They all fulfilled different functions: A shader program for the graphics card, a Processing sketch for generating visual output, the computational proof of a mathematical theorem and data processing - were some of the tasks depicted in the algorithms.
The printed source codes were distributed at the bar for participants to look at them for inspiration.

All 21 code poems that were written on that evening can be found in the appendix (10.4 and 10.5)

6.2. Invitation

The event was advertised on the social media platform Facebook by me and by Fabian Frei in two separate events. In sum, 60 accounts were listed as possible participants. At the event which lasted about 4 hours there were around 40 - 50 people.

For while { Exception println #update let !doctype @type } ;
Write Code Poetry like the Old Masters!

May, 16 2019
7pm - 11 pm
Dokapi, Domgasse 2, Art University Linz

With food, beer and cake

for, while, {, Exception!, ||, println, #update, let, !DOCTYPE, @type, }, ;

No idea what that means?
Perfect!
If you also like to play with words, write poems or short stories, this is the place to be and where you'll become a true code poet.

Think about "Writing in the style of Shakespear." ir "Painting in the style of Frida Kahlo" - here it is "Poetry in the style of Code".

We will write code together without thinking about how or if it works. It's about style. How does a poem look like if it's written in the style of Java, HTML or Python?

Example:

```
class love {};
void main(){
    throw love();
}
```

How?
We meet and through typical source code and pick the stuff that we like from it: sentence structure, phrases, rhythms..
Then we start writing poems and try to stick to the design vocabulary.

Why?
I am writing my thesis on the influence of code-logic on everyday life. Thereby I stumbled upon code poetry and some poets who apply computational knowledge in creative writing. That's what made me especially interested in the cross-over.

-> -> -> -> The whole thing is part of an evening of freestyle poetry writing hosted by my colleague Fabian Frei who will feed a neural net with your poems.

6.2. Participants

Most of the 40 - 50 participants were students of Art University Linz of different and study fields: Interface Cultures, Visual Communication, Experimental Design, Space Design Strategies. There were also former students, teachers, students of the scientific Johannes Kepler University Linz and writers.
The invitation was especially directed to people who have never coded before. Programmers were invited, showed up and contributed as well, but my research was more about style than about function: How would non-coders work with “Code” as a genre of writing and translate it into their own texts?

6.3. Ways of Analysing a the Code Poems

Comparison with functional Source Code

Some participants who were new to programming read one of the example codes and then composed their own poem. In this the syntax of a certain programming language is still recognizable, but the program is no longer executable.

Example: Comparison of a shader program written in GLSL with a code poem.

1. Source Code:

```
attribute vec3 a_position;
attribute vec3 a_normal;
attribute vec2 a_texCoord;
uniform mat4 u_modelView;
uniform mat3 u_normalMatrix;
uniform mat4 u_projection;
uniform mat4 u_invView;
uniform vec3 u_lightPos;
uniform mat4 u_eyeToLightMatrix;
varying vec3 v_normalVec;
varying vec3 v_lightVec;
varying vec3 v_eyeVec;
varying vec2 v_texCoord;
varying vec4 v_shadowMapTexCoord;

void main() {
      vec4 eyePosition = u_modelView * vec4(a_position,1);
      v_normalVec = u_normalMatrix * a_normal;
      v_eyeVec = -eyePosition.xyz;
      v_lightVec = u_lightPos - eyePosition.xyz;
      v_shadowMapTexCoord = u_eyeToLightMatrix*eyePosition;
      v_texCoord = a_texCoord;
      gl_Position = u_projection * eyePosition;
}
```

2. Code Poem

```
Highlighter
If you want to light up,
you will need to find a_position;
YOu will not look like a_normal;
cos you done it using shadowMapTexCoord;
Mat4 the colour that fits
I you do not smooth(); out the
t.highlight = false;
```

```
Now put yourself in the right
u_lightPos;
and you got the u_modelView;
```

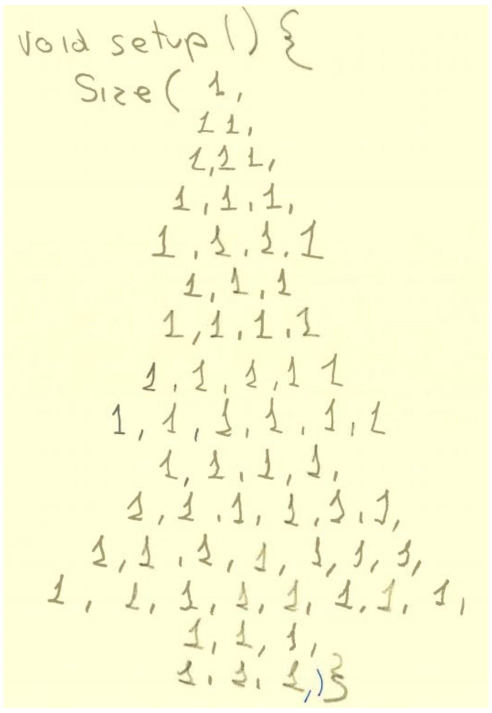
Code Structure and Visual Poetry

Indentation, bracketing, use of ASCII symbols and line breaks are all are functional in an executable program but can be used as style devices.

Example: Symbols for code clusters and comments as a dstyle device

```
{
        {
                {
                }
        }
}
//////////////////////////////////
```

Examples: Visual Code Poetry using Indentation



```
void setup() {
      Size( 1,
              1 1,
              1,1 1,
             1 , 1 , 1,
            1 , 1 , 1 , 1
              1, 1 , 1
            1, 1 , 1 , 1
           1 , 1 , 1 , 1 1
        1 , 1 , 1 , 1 , 1 , 1
            1, 1 , 1 , 1,
         1 , 1 , 1, 1 , 1 , 1,
       1, 1 , 1 , 1, 1, 1, 1,
   1 , 1, 1, 1, 1, 1, 1, 1, 1,
              1, 1, 1,
              1 , 1 , 1 ,)}
```

Fig 13: Code Poem by Cesar Escudero Andaluz

Executable Code Poetry

One of the beauties of merging poetry with code lies in the representation of complex life problems or questions as equations that can be simply run by a computer and thereby solved.

Example: Executable Code Poem written in the Java Programming Language

```
class ThrowLove{
     static void give(){
          try{
               throw new Love("whui");
          }
          catch(Love romantic){
               System.out.println("Caught inside ThrowLove");
          }
     }
}
```

Example: Code Poem by the Author (2014)

```
Art createBestPiece(){
     Art masterpiece = create();
return masterpiece;
}
```

This overview just gives a little insight into the ways how code poems could be analysed. Code obfuscation as mentioned in chapter 4.6. is an approach that could be compared to Dada.
Another genre of code poetry are those, where the code and the output of the executed algorithm are both part of a poem.
This thesis aims to look at code and poetry as part of language on a broader scale and therefore does not go into further detail with these strategies.

7. Conclusion

In the beginning I invited all readers to join my exploration of logic and art. We started by a definition of code and poetry and a comparison of the two. They lead us to the expression of logic and creativity through language. By defining computation and simulation, the philosophical area of epistemology was reached: what can we know? Wittgenstein helped a lot in this part of the thesis.
On the one hand, he aimed to explain all there is by one unifying language, on the other he acknowledges that there are as many possible explanations of the world as there are individual languages. By today, all the programming languages can be added to this spectrum.

Each language is more or less compatible with the others and through all of them we converse and try to create a coherent image of our collective reality. Joscha Bach is quoted in chapter 2 arguing that computer science might be the best way to explain how the world and the mind work because according to him this computer science has a tool that philosophy has not: computation.
At this point, it was important not to fall into the trap of believing that reality is a computable simulation that runs in the neocortex of a primate. The history of mathematics shows that some very intelligent people like Gottlob Frege and Bertrand Russel spent a good part of their lives only to fit the fact based-world of math into one explanation.

Still, if everything could be explainable on a computational level, we would have a nice bridge to code and poetry again: Running code on a machine and “executing” a poem on someone's mind would then become the same thing.

By the way, “same things”: Thanks to Gödel we learn to understand that a system can never be complete because it does not contain rules about itself. That makes our reality incomplete by definition, some questions have to stay unanswered.
Simulations help, not to solve this problem but to observe it: If we ask a questions where the answer must lie outside of it, the machine that runs the simulation will break - and we find ourselves in the reality where the simulation was started from. In order to not understand this thoroughly, the reader may continue reading 800 pages of “Gödel, Escher, Bach” by Douglas R. Hofstadter.

At least we know what we are not: Each attempt to create an artificial mind or an artificial world will give us another answer that we can exclude from the list of what we possibly are. Even if there was a conscious robot that could question its existence -indistinguishable from a human -, the answer to the question “Why do I exist?” would be different to that of a human.

The next trap would be to not move further and continue feeling insignificant by these questions that can not be answered. That is where the exploration turns back to poetry. It is the art of playing with language, of subverting its meaning,taking it apart and assembling it again.

Both in logic and poetry there are the same games: Self-Reference, Repetition, Infinity, Recursion, Subversion, Paradox and Nothingness - when applied to dull reasoning, they will inspire the creative linguist, poet or programmer.

In chapter 4 art becomes the focus of the exploration. How to break out of reality? How to create it while living in it? How to construct the smallest self-referential loop that can still be observed as a loop, rather than a still image? Where in language does meaning end and form start?
These projects are especially joyful to explore after the readers familiarize themselves with the tradition of deterministic reasoning and the way language works.

In the end of this exploration there are many ways to proceed: A performative lecture on the game of logic and poetry and the creation and analysis of code poems are two ways that I chose to go.
The questions raised by pursuing any of them could not have been asked without the research that is summarized in this thesis - which is the best outcome me, as the author could have hoped for.

8. Bibliography

Abelson, H., Sussman, G. J., & Sussman, J. (1996).Structure and interpretation of computer programs (2nd edition).

Anderson, V. (2016). Fluxus : Event Scores and Their Performance.

Bach, J. (2013). How to build a Mind. 30c3 [video file]

Bach, J. (2018). The Ghost in the Machine An Artificial Intelligence Perspective on the Soul. 35c3. [video file]

Bergstrom, K. (2010). DIY Useless Machine. [video file]

Bertran, I. (2012). code {poems}.

Bezhanishvili, G., & Fussner, W. (2013). An Introduction to Symbolic Logic.

Biletzki, A., & Matar, A. (2018). Ludwig Wittgenstein. In The Stanford Encyclopedia of Philosophy.

Calvino, I. (1962). The Nonexistent Knight.

Cellucci, C. (1992). Gödel's Incompleteness Theorem and the Philosophy of Open Systems.

Church, A., & Turing, A. M. (1937). On Computable Numbers, with an Application to the Entscheidungsproblem.

Croll, A. (2017). Code and Poetry, a Conversation.

Cryan, D. (2016). Logik.

Hesselström, K., & Aslund, J. (2001). The Shakespeare Programming Language. [website]

Hofstadter, D. R. (1980). Gödel, Escher, Bach: An Eternal Golden Braid (1st ed.).

Kagan, M., & Conrad, D. (2014). Writing Code As Poetry; Poetry as Code.

Krauss, R., & Chiu, C.-Y. (1997). Language and Social Behavior.

Krebsbach, K. D. (n.d.). Computer Science: Not about Computers, Not Science.

Lage, M. (2013). First Stanford code poetry slam reveals the literary side of computer code.

Launay, M. (2016). It All Adds Up: The Story of People and Mathematics.

Leivant, D. (1994). Higher Order Logic.

Linsky, B. (2019). Principia Mathematica. In The Stanford Encyclopedia of Philosophy.

Mataes, M., & Montford, N. (2005). A Box, Darkly: Obfuscation, Weird Languages, and Code Aesthetics.

Maurer, E., Schürgers, N. J., Demmerling, C., & Gönner, G. (2015). Metzler Philosophen-Lexikon.

McGuinnes, B. F. (1979). Ludwig Wittgenstein and the Vienna Circle: Conversations Recorded by Friedrich Waismann.

Mischer, K., & Traxler, T. (2011). Mischer’Traxler Studio. [website]

Mühlbacher, J. R. (2009). Betriebssysteme. Grundlagen. Schriftenreihe Informatik.

O’Brien, L. (2015). Code Watch: The best programming book of the decade.

Queneau, R., Eco, U., & Calvino, I. (1958). Exercises in Style.

Solon, O. (2012). Designer publishes collection of developer-written “code poems.”

Street, T. (2015). Arabic and Islamic Philosophy of Language and Logic. In The Stanford Encyclopedia of Philosophy.

Temkin, D. (2011). Esoteric Codes.

Trostel, S. (2018). All Creatures Welcome. [video file]

Vee, A. (2017). Coding Literacy: How Computer Programming is Changing Writing. In Software Studies.

Videira Lopes, C. (2014). Exercises in Programming Style.

Waner, S., & Costenoble, S. (1996). Predicate Calculus.

Wittgenstein, L., Russell, B., & Ogden, C. K. (1922). Tractatus Logico-Philosophicus.

9. Image References

Fig 1:
Last lines of a code poem by Álvaro Matías Wong Díaz written in the programming language Java from the book “code {poems}” by Ishac Bertran (2013)

Fig 2:
Code poem by Paul Illingworth written in the programming language Java from the book “code {poems}” by Ishac Bertran (2013)

Fig 3: Code poem “wonderer.wander” by the author

Fig 4 1- 6::
Illustration by the author of infinity, self-reference, recursion, repetition, paradox and nothingness.

Fig 5:
Still frame from the film “Hybris” showing actor Bart van der Schaaf and director Arjan Brentjes

Fig 6: Bergstrom, K. (2010). DIY Useless Machine. Retrieved from https://vimeo.com/34453539

Fig 7: “Collective Works” (2011) by Studio Mischer’Traxler

Fig 8: Documentation of “FURTHER, FURTHER, FURTHER” at Gallery BB15, June 15 2019

Fig 9 1-13:
All Rounds and all Stops of “FURTHER, FURTHER; FURTHER”, June 15 2019, Gallery BB15 Linz

Fig 10: Illustration of semiotics of a guided tour by the author

Fig 11:
Illustration of the imaginary artwork described during the event "FURTHER; FURTHER; FURTHER" on June 15 2019, at Gallery BB15 in Linz

Fig 12: Participants at the poetry evening at Art University Linz, May 16, 2019

Fig 13: Code Poem by Cesar Escudero Andaluz

10. Appendix

10.1. Texts about “Variable” by Maja Burja and Urška Aplinc

Any talk about this work is limited to the content of the work itself. The material used in the performance is never used for applications. We share parts of texts that were written by others and photo documentation we had no control over.
»Self-reflexivity is the central drive of Maja Burja and Urška Aplinc’s work Variable. With each new repetition, the performative work is extended in accordance with the scope of the audience’s responses, which are the main material of the work’s content. The work is completely fragile in its dependence on the art audience’s interest, for which and through which it is performed; at the same time, it is also a ruthless machine for the neutralisation of its interpretive power and freedom. Although in constant risk of reproducing the system, it must not remain silent…« (Tjaša Pogačar in Vladimir Vidmar, 2017)
»Variable is a performance by Maja Burja and Urška Aplinc conceived as an autopoietic, permanently evolving work of art. The authors use text and photographs as basis for a debate with the audience, which is then recorded and added to the body of work used in the next incarnation of the performance. Autopoiesis is achieved through Deleuzian repetition, ensuring that every new incarnation irreversibly changes the content of the work. Inclusion of the audience in the performance brings it closer to action art and introduces a certain “autopoiesis of interpretation” in which the participants explore their perception of the perception of those before them ad infinitum. Thus in Variable the history of subjective, individual interpretations of a work of art become the work of art itself. «
(Bojan Stefanović, 2016)

»For the sake of precision, I asked the artists to give me a description of the performance, but they insist on the performativity of the work – I remembered what I did, even though it may be inaccurate and inadequate. [...] Thinking about the debate, which subsequently developed with approximately ten participants, may be a key moment for the interpretation of the work: we discussed chance, fragmentation, the connection between content and meaning, fiction and facts and interpreting in vain. We wondered what we - as participants of the performance - can do; is there anything that we should do? [...] I can think about it with the help of the concept of Relational Aesthetics by Nicolas Bourriaud, but with a certain level of restraint: the situation was not focused on building relationships between us; how we said things was not in the forefront, but rather (still) what we said. I did not sense the intention to expose the rigid, artificial and self-referential nature of the art system in the performance; the constant evasiveness of the subject matter (essence) was also surprisingly not something that would appear hopeless or annoying, but rather seemed to add to the charm of our collective effort to make a myth out of the smallest gesture. «
Simona Žvanut, 2016)

10.2. Interview with Maja Burja and Urška Aplinc about their work "Variable"

Can you describe the underlying "algorithm" of the event? How did you "execute" the program? Meaning how did you guide the audience through the performance?

U: The performance now consists of different parts - some pertaining to documentation and some to the responses of the audience in present time, but the initial structure differed and in part developed through time.

M: In principle, we don't talk about this work outside of the performance, as we try to exclude ourselves from the work in as many ways as possible. What arises firstly as an answer to this question is the uncertainty of the relationship between the algorithm and the work - to what extent the algorithm determines the work itself. Would it be possible to skim the work of the accumulated iterations and unveil it as a mere algorithm, or would we perhaps notice that the algorithm itself was established and modified in parallel with the development of the work, and was an algorithm only to the extent that the implementation was consistent and is outlined subsequently? We can look at the issue of guidance from a similar point of view.

What was the motivation for the project and did the outcome differ from you expectations?

U: I don't think we can talk about motivation regarding this project. I link motivation to intent, while this work was perhaps shaped more by restrictions and necessities, fueled by chance and transient construction of meaning.

M: After a few repetitions of the work, I found an interesting scheme in my notebook (subsequent identification): words will become a thing // quoting and visualizing organized around a newly created core of the work // 1,2,3 → → → a description of non-existent works → text → reader → word → listener → description → viewer → image → thing --> an existent work. With this "coincidence" I wish to explain my attitude towards motivation or expectations and similar concepts. Such links are not linear and are often ambiguous. A work can often open up more issues than it clears up.
// I was contemplating the passage from The Nonexistent Knight (Calvino) with similar reservations a while ago.

What would be a possible different outcome if it was repeated? What are the variables that let the outcome change?

U: The outcome has been changing with each repetition, as the work allows for the inclusion of different instances of outside, external circumstances and contents. The variables change with each repetition and so does the content.

M: The work has been performed many times, though it is hard to say how many times it has been repeated - it depends on how we understand repetition. The title of the work is Variable and so is its content, would be the poetic answer, while the variables of the work are perhaps the same as with any and all other works and the illumination of these variables could be viewed as one of the accomplishments of this work.

“Thus in Variable the history of subjective, individual interpretations of a work of art become the work of art itself. “
Where does self-interpretation lead? Where could the constant repetition of the interaction with the audience and constant referral to the starting point create extra-value?

U: I don't know the answer to this question, perhaps because all the while referring to the starting point, the performance is obscuring its own history and making it incomprehensible at the same time. Where does any work of art create extra value?

M: One time, while talking about a different work of mine, a professor asked me why I had exchanged my place/role with others - where I thought a surplus of value would be created. I can't remember what I answered, but for me it was more important to remember this question, as it is a question which has many more particular answers that change in time, rather than one driving force or comprehensive answer. I think that there’s a process, parallel to this question, at work in Variable.

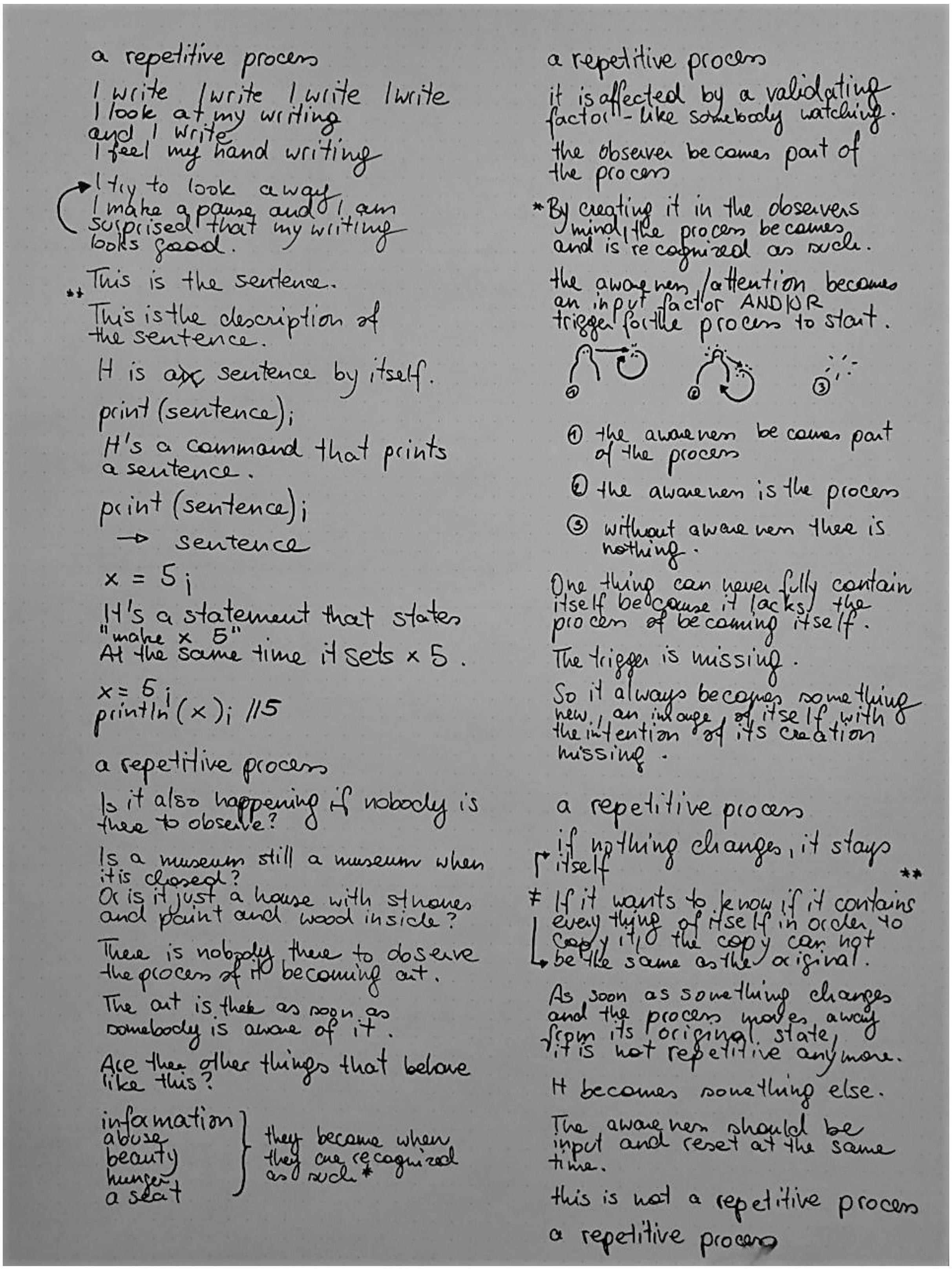
What’s the nature of art that this performance results in?

M: This is, after all, a question that the work, the participants and the artists are posing among others. I'm now thinking about the figure of speech "the nature (of something)". It prompts me to think about the question of the nature of memory, which I could use as a metaphor for this work. Memory is an individual, but also a collective practice, subjugated to digression, repressions and distortion, closely related to the dynamic and continuous production of archives, dictated by technological and cultural mediation and can be interpreted only through discursive and material phenomena, which, in turn, are reinvented in each interpretation process again and again in the dynamics of the aforementioned.

Could a similar process be achieved without an audience, just conducted as individual artistic practice?

M: The question that arises here is what ‘’an individual artistic’’ practice might be. I'm thinking here in line with the institutional art theory, which says that art is only what is part of the institution. The institution of art, however, is internalized, embodied in individuals (conceptual and perceptual frameworks). Does it make sense then to build a work or gesture (of an artist) from the recognition of the impossibility of separation from society, from the institution(al) ... to accept the otherness /foreignness of one's own gesture and accept the infinite appropriate and interpretive power of art, which is merely performed through manifestations that are potentially (materially and discursively) infinite? Nevertheless, being significantly influenced by the outside does not mean that we are relieved of our influence or responsibility; unfortunately, creating a work which is not one way or another ‘’my own’’ is still something I haven't been able to accomplish and what drives my further endeavours. (Online Questionnaire, July 30th, 2019)

10.3. Text “A Repetitive Process” by the Author



a repetitive process

I write I write I write I write
I look at my writing
and I write
I feel my hand writing

I try to look away
I make a pause and I am
surprised that my writing
looks good.

This is the sentence.

** This is the description of
the sentence.

It is a~~x~~ sentence by itself.

print (sentence);

It's a command that prints
a sentence.

print (sentence);
→ sentence

x = 5;

It's a statement that states
"make x 5"
At the same time it sets x 5.

x = 5;
println (x); //5

a repetitive process

Is it also happening if nobody is
there to observe?

Is a museum still a museum when
it is closed?
Or is it just a house with stones
and paint and wood inside?

There is nobody there to observe
the process of it becoming art.

The art is there as soon as
somebody is aware of it.

Are there other things that behave
like this?

information
abuse
beauty
hunger
a seat

} they become when
they are recognized
as such *

a repetitive process

it is affected by a validating
factor - like somebody watching.

the observer becomes part of
the process

* By creating it in the observers
mind, the process becomes
and is recognized as such.

the awareness / attention becomes
an input factor AND/OR
trigger for the process to start.

① ② ③

① the awareness becomes part
of the process

② the awareness is the process

③ without awareness there is
nothing.

One thing can never fully contain
itself because it lacks the
process of becoming itself.

The trigger is missing.

So it always becomes something
new, an image of itself with
the intention of its creation
missing.

a repetitive process

if nothing changes, it stays
itself **

* If it wants to know if it contains
every thing of itself in order to
copy it, the copy can not
be the same as the original.

As soon as something changes
and the process moves away
from its original state,
it is not repetitive anymore.

It becomes something else.

The awareness should be
input and reset at the same
time.

this is not a repetitive process

a repetitive process

10.4. Anonymous Code Poems

21 code poems from the event “For while { Exception println #update let !doctype @type } ;” on May 16th 2019 at Art University Linz, conducted by the author, composed by participants of the event.

```
1.
Write a Poem
I give instructions to myself
To start doing.
Think of beautiful words.
Now I hae them.
WHere to place them.
I don’t know the meaning of
```

```
2.
Being safe in a room.
Standing alone, not lonely.
I just care about eating my
Risotto.
I’m not a poet.
```

```
3.
The relationship
inBegin, you typename
The RandomGenerator return to
be
A RandomSample-Included
<castlib> which outEnd
The fact that his genitals
Had size_t
```

```
4.
#undefined
!can.produceiii
```

```
5.
PointAndClick.PointAndClick.
PointAndClick.          (point)
                I love it.
                        (click)
int stein = 1;
string bert = “a”;

for (ScientistGenerator){
     return bert + stein;
     stein++
     bert ++
}
```

```
6.
{
     {
          {
          }
     }
}
//////////////////////////////
```

```
7.
imagine(child){

this.children.dream(child);
     for(int i = 0; i<4; i++){

this.children.splice(i,26);
     }
     return i = null;
};
```

```
8.
Class Color{
      Color c;
      Color (color c){
      this.c = c;
      }
}
```

```
9.
Knit one, pearl one,
Print line, send;
At_the_terminus, my wooly
friend.
```

```
10.
Com_pu_ter
Komp_oo_te
```

```
11.
Seek meaning
Find void
Execute infinite
searchalgorithm
Enhance Selfworth
```

```
12.
Class Love{
      Love love;
      Love (Love  love){
      This.love = love;
      }
}
Love love = new Love(love);
```

```
13.
the test
Friday night
The mChild called boolean
isMarked;
His mParent found out;
The public static final class;
```

```
even the mPriority = priority;
the child got mMin = 0;
value;
double value;
trouble;
boolean isEmpty.
```

```
14.
if(urlove)>5;
breakheart(#34856);
repeat;

void setup(){
      size(640, 360);
}

void draw(){
      background(127);
      yourmum(fat);
}
```

```
15.
This poem does not
compile.
Sad smiley is missing
on line 3, char 27.
```

```
16.
Highlighter
If you want to light up,
you will need to find
a_position;
YOu will not look like
a_normal;
cos you done it using
shadowMapTexCoord;
Mat4 the colour that fits
I you do not smooth(); out the
t.highlight = false;
Now put yourself in the right
u_lightPos;
and you got the u_modelView;
```

```
17.
class ThrowLove{
     static void give(){
          try{
               throw new Love("whui");
          }
          catch(Love romantic){
               System.out.println("Caught inside ThrowLove");
          }
     }
}
```

```
18.
add.not(not_to_add).calculate as -> if{
     careful.add(not:to.ad) -> i;
     fearful = character.leave();
     to >> void_and_beyond = terrific;
     can !undo;
}
care = less.than(option(1));
to_put >> down;
would throw = up;
!put not >> on me.you();
in 1 attempt_of = perfection;
if(I_turn){
     you.turntoo;
}
```

```
19.
template(<typenameLifeChanges,typename Spontaneous)
     LifeChanges Random (houses, Cats, boyfriends){
          }
     }
```

```
20.
attribute  human          a_firstlove;
attribute  human          a_hiswife;
attribute  animal         a_greyhounddog;
place      recreation     p_thelocalpark;
```

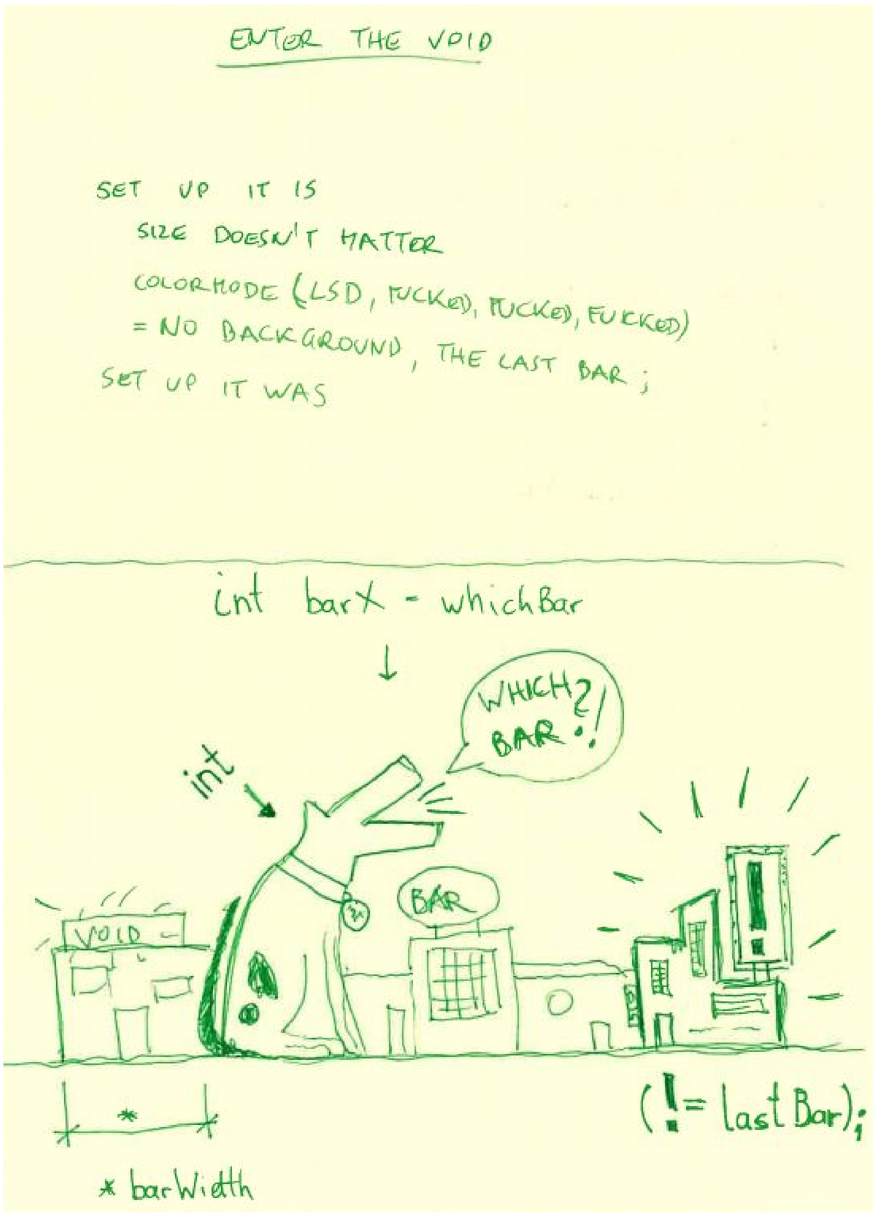
```
feelings   confusion        f_whattosay;
feelings   confidence       f_lackof;
feelings   glamour          f-betterhair;
words      excuse           w_callafriend;
words      unspoken         w_myheartstillhurts;
reality    improvement      r_lifeismuchbetter;

void main(){
     if(p_thelocalpark){
          while(a_greyhounddog.runs){
               throwball();
               checkphone();
          }
     if(a_greyhounddog.barks){
          if(lookup()==a_firstlove){
               for(i=5;f_whattosay;i--){
                    say(w_callafriend);
                    think(w_myheartstillhurts);
                    know(r_lifeismuchbetter);
               }
          waveto(a_hiswife);
     }
}
```

```
     function noFunction(void nothing){
          super(noThing);
          let nothing = " ";
          return nothing;
     }
```

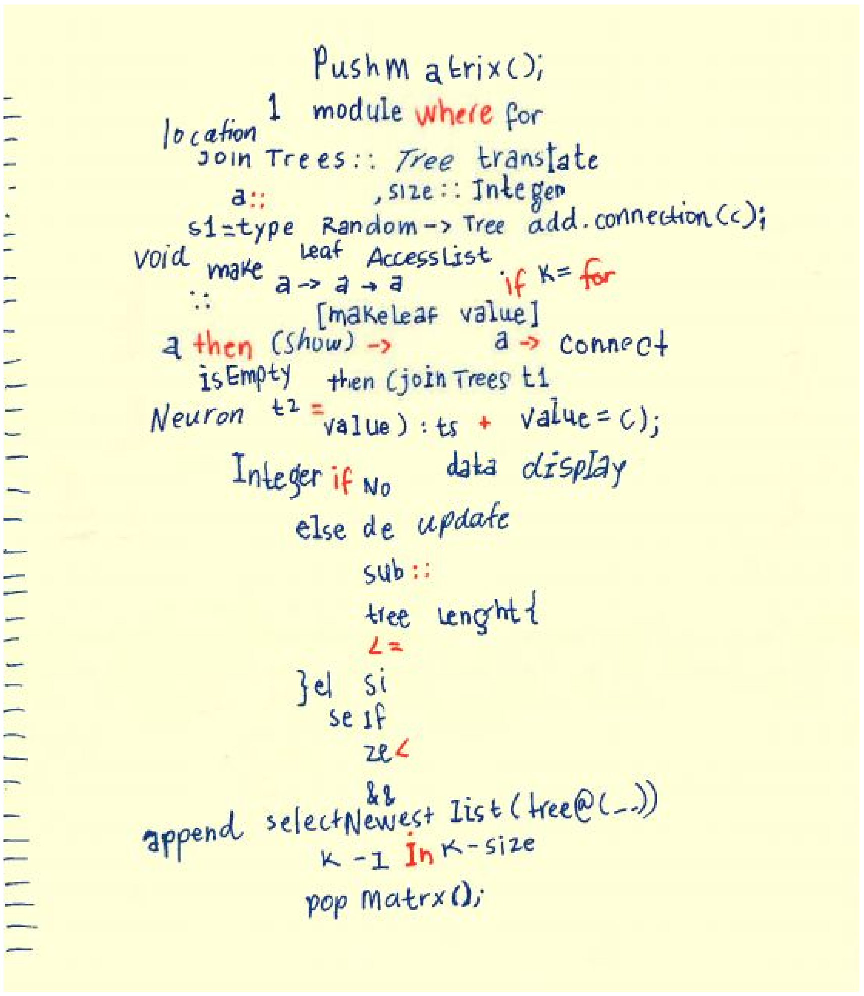
```
21.
the r=random(8) dance
a smooth();
lightning stroke(255); floats
t !=other faces
you start t.move();
in(frameRate) = 2000;
you feel the void
fill(100); percent of your body
totalThings start to appear.in your head
nowhere else you can a.add
this thing in your ellipse(x,y,r,r);
```

10.5. Visual Code Poems

6 visual code poems from the event “For while { Exception println #update let !doctype @type } ;” on May 16th 2019 at Art University Linz, conducted by the author, composed by participants of the event.

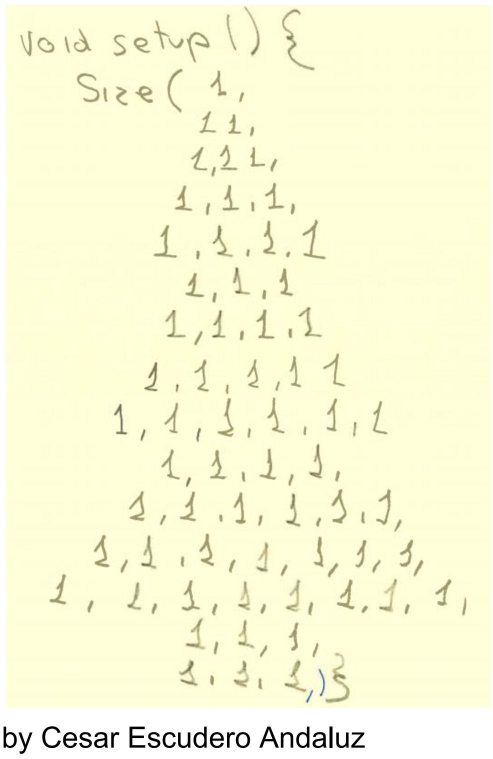


by Gabriel Rosza

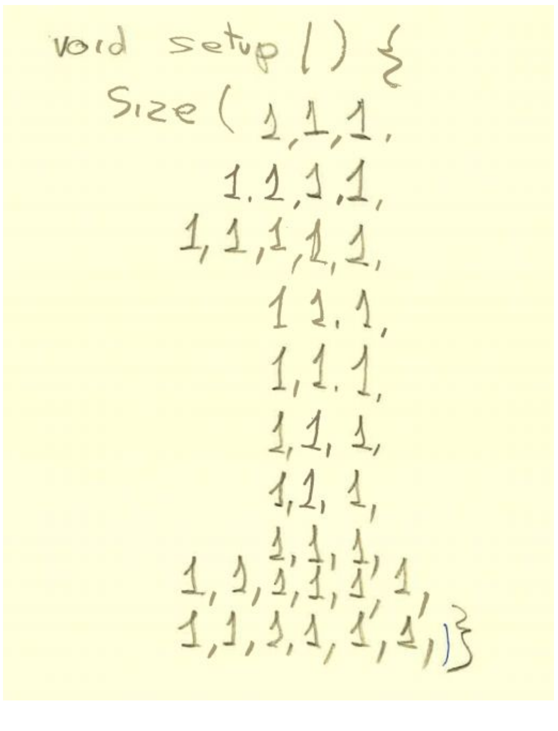


```
                 Pushm atrix();
              1  module where for
      location
        Join Trees:: Tree translate
           a::        ,size:: Integer
        s1=type  Random -> Tree  add.connection(c);
  void  make   leaf  AccessList
      ::       a -> a -> a      if K= for
                 [makeLeaf value]
     a then (show) ->       a -> connect
        isEmpty   then (join Trees t1
    Neuron   t2 =value) : ts  +  value = c);
           Integer if No    data display
                else de update
                     sub::
                     tree  lenght{
                     <=
                }el  si
                  se if
                     ze<
                     &&
  append  selectNewest list(tree@(_.))
                 k -1 In k-size
                pop Matrx();
```

by Gabriella Gordill

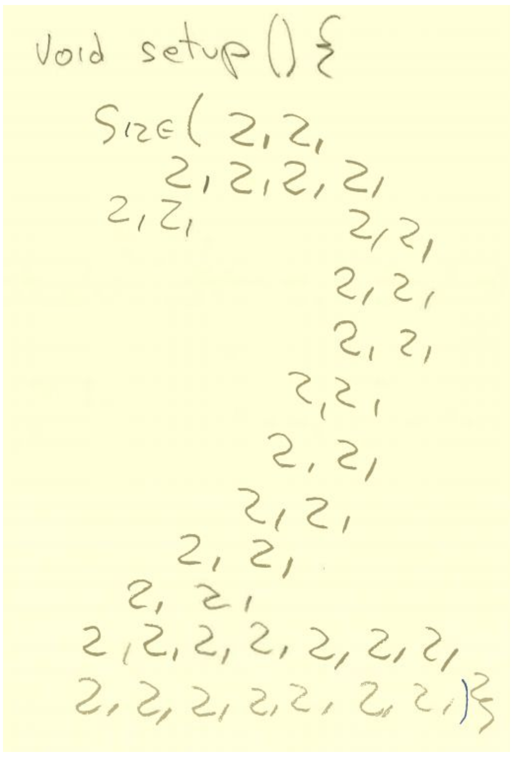


```
void setup() {
   Size( 1,
        1 1,
       1,1 1,
      1,1,1,
    1,1,1.1
      1,1,1
     1,1,1,1
    1,1,1,1 1
  1,1,1,1,1,1
     1,1,1,1,
   1,1 .1,1,1,1,
  1,1,1,1,1,1,1,
1, 1,1,1,1,1.1, 1,
     1,1,1,
     1.1.1,)}
```

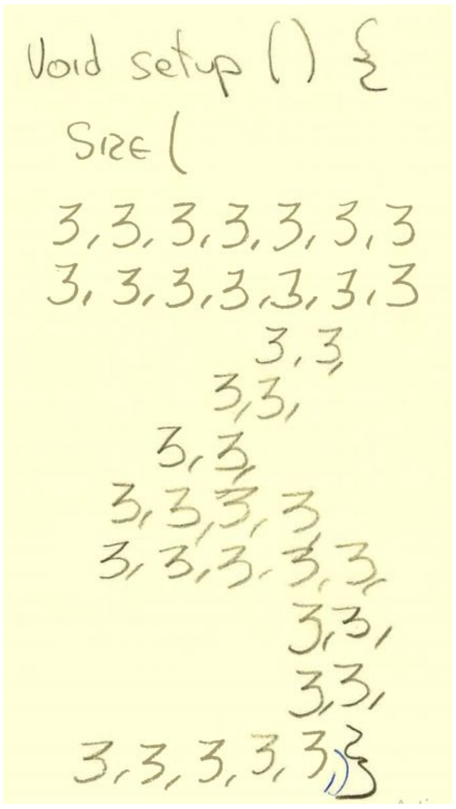


```
void setup() {
   Size( 1,1,1,
        1.1,1,1,
      1,1,1,1,1,
          1 1.1,
          1,1.1,
          1,1, 1,
          1,1, 1,
          1,1,1,
      1,1,1,1,1,1,
      1,1,1,1,1,1,)}
```

by Cesar Escudero Andaluz



```
Void setup() {
    Size( 2,2,
        2,2,2,2,
     2,2          2,2,
                 2,2,
                 2,2,
              2,2,
             2,2,
           2,2,
        2,2,
      2,2,
    2,2,2,2,2,2,2,
    2,2,2,2,2,2,2,)}
```



```
Void setup() {
  Size(
  3,3,3,3,3,3,3
  3,3,3,3,3,3,3
              3,3,
            3,3,
         3,3,
       3,3,3,3
      3,3,3,3,3,
               3,3,
               3,3,
    3,3,3,3,3,)}
```

by Cesar Escudero Andaluz