University of Zurich[UZH]

# Agent-based Financial Economics
# Lesson 1: The Hermit

Luzius Meisser, Prof. Thorsten Hens

luzius@meissereconomics.com

"What I cannot create, I do not understand."

- Richard Feynman


Ivane Goliadze

# About this course

- All information on http://course.meissereconomics.com
- Every Friday from 14:00 to 16:00 at KOL-F-123
  → To do: define when to do the breaks
- 14 Lessons, last two for presentations
- We will form teams of two during the break
- Unique setup with custom-made software
- Course is completely new
- Modelling ideas borrowed from “Economic Foundations of Finance” by Thorsten Hens and Helga Fehr-Duda, 6 ECTS

# Typical lesson structure

- Discussing last week's exercise
- Famous models
- Theory, methodological background, maths
- Extending our model, what do we expect?
- Next exercise, see what actually happens

# Structure Today

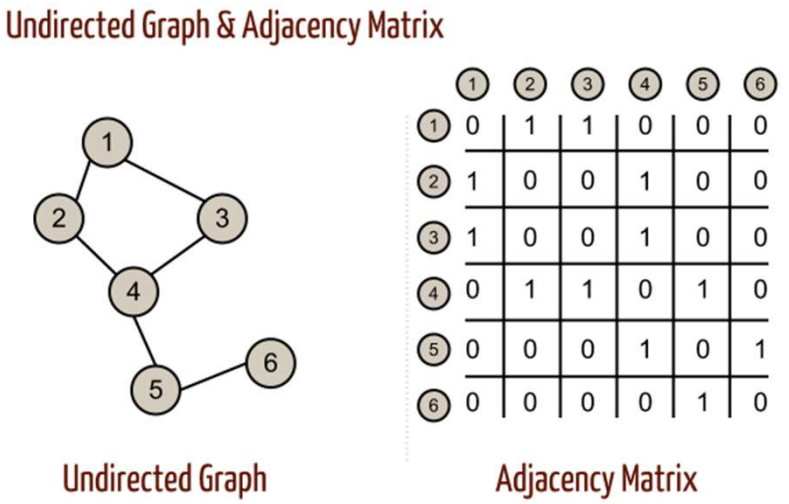
- About this course
- Theory: Chaos and complexity
- Famous model: Shellings segregation game
- Break: forming of teams
- Economic basics: utility and production
- Software engineering: OO, Java, Eclipse, Git
- General setup and first exercise

# About me

- Studied Computer Science at ETH, 2006
- Co-founded and sold secure cloud storage startup
- Fintech investor, board member of Bitcoin Association Switzerland
- Teaching of object-oriented programming at FHNW
- Studied Economics at UZH, 2016
- Currently pursuing a PhD on “Agent-based financial economics”
- I believe that there is a huge untapped potential of software engineering in economics and finance.

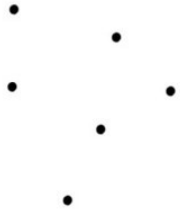
# Philosophy

- Agent-based vs equation-based
- Local vs global decisions
- Invisible hand vs central planning
- Decentralized vs holistic
- Object-oriented vs data-driven
- Graph vs matrix
- Modular vs monolithic
- Divide et impera vs aggregation

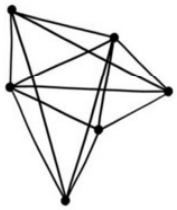



You can always look at a coin from two sides. Nonetheless, it keeps being a coin.

# TOOLS OF A SYSTEM THINKER



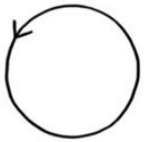
DISCONNECTION



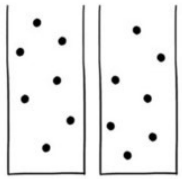
INTERCONNECTEDNESS



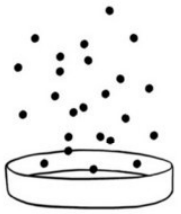
LINEAR



CIRCULAR

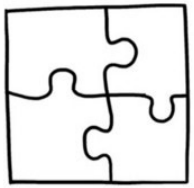


SILOS

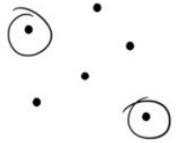


EMERGENCE

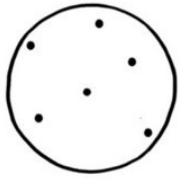
PARTS



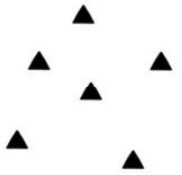
WHOLES



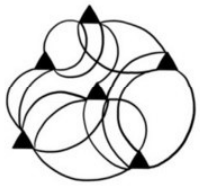
ANALYSIS



SYNTHESIS



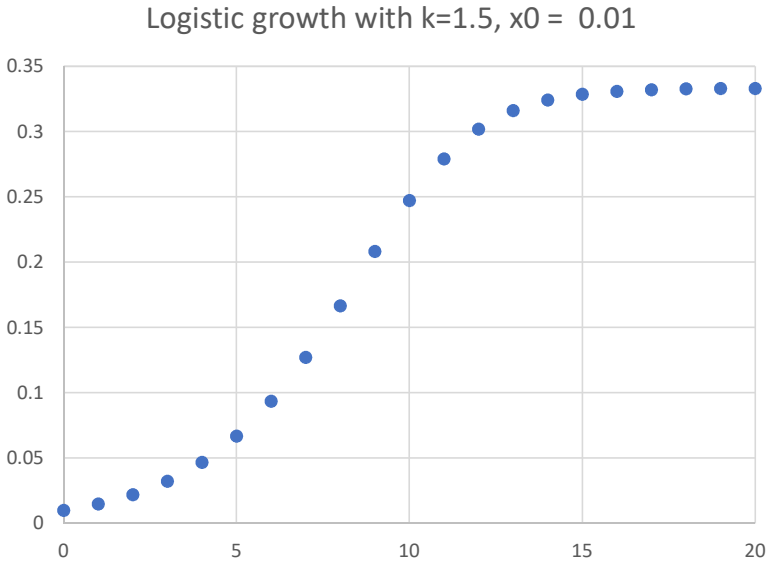
ISOLATION

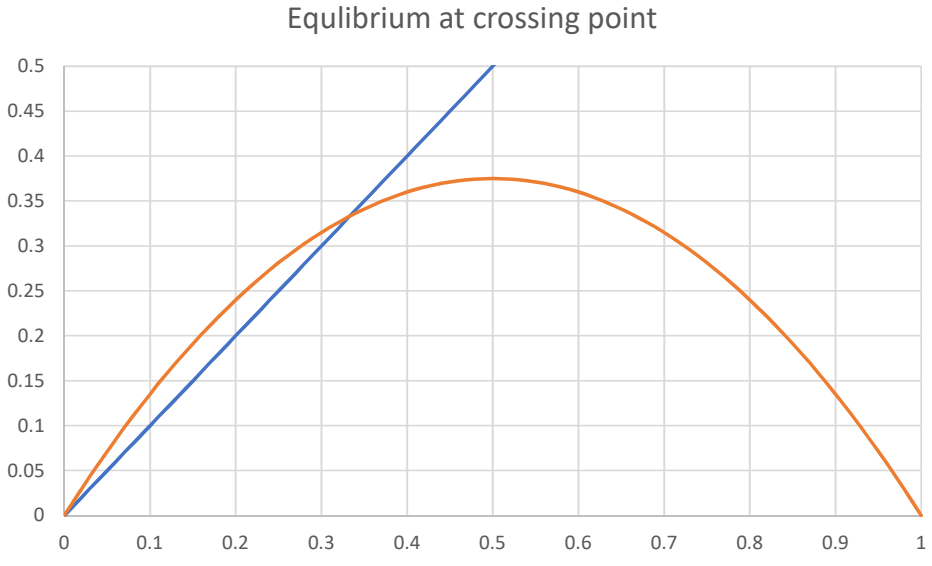


RELATIONSHIPS

# Chaos and Complexity

Classic example: discrete logistic growth. $x_{k+1} = r x_k (1 - x_k)$



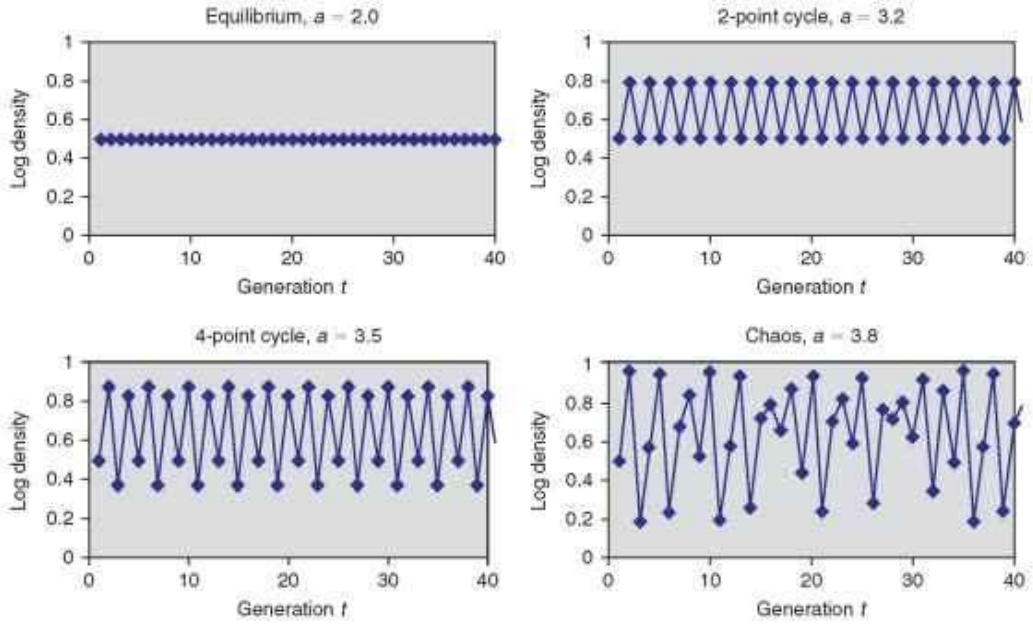
# Chaos and Complexity

## Simple systems can exhibit chaotic behavior.

Classic example: discrete logistic growth

$$x_{k+1} = r x_k (1 - x_k)$$

For a model in which prices behave like this, see: Cars Hommes, «Behavioral Rationality and Heterogenous Expectations in Complex Economic Systems», page 14



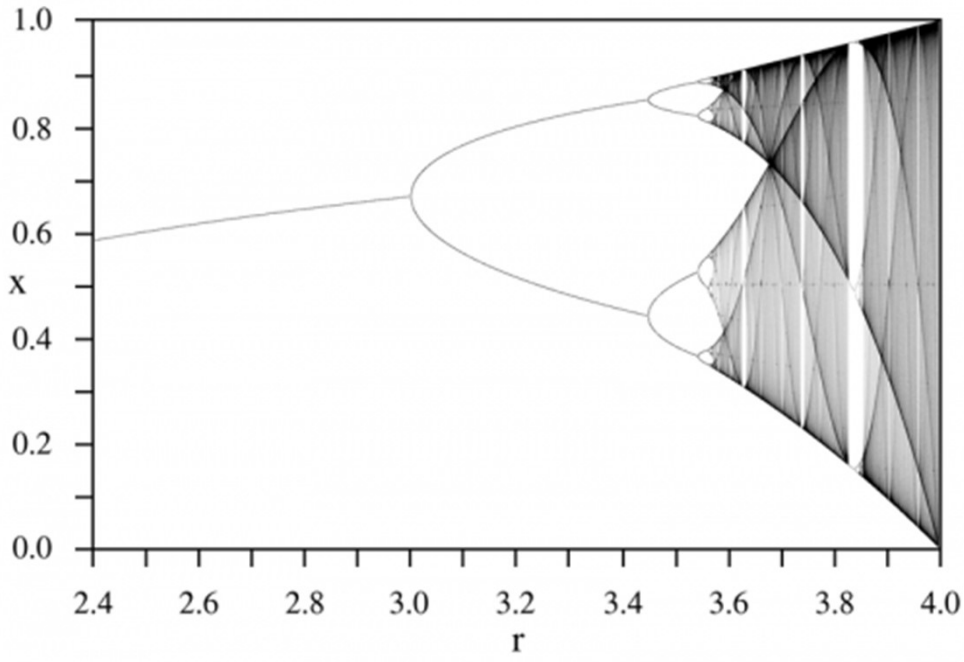
# Chaos and Complexity

Classic example: logistic map

$$x_{k+1} = r x_k (1 - x_k)$$

See also: Cars Hommes, «Behavioral Rationality and Heterogenous Expectations in Complex Economic Systems», page 14

→ Systems get chaotic very quickly. In chaotic systems, small arbitrary assumptions or errors can make a big difference.
→ Results are worthless without having a stable, known case as an anchor point.



# Chaos and Complexity

The magnetic pendulum: classic example of a continuous system that exhibits complexity.

Multiple equilibria, practically impossible to tell at which one it will end up.

→ To avoid chaos, we should prefer systems in which small errors are not allowed to add up.

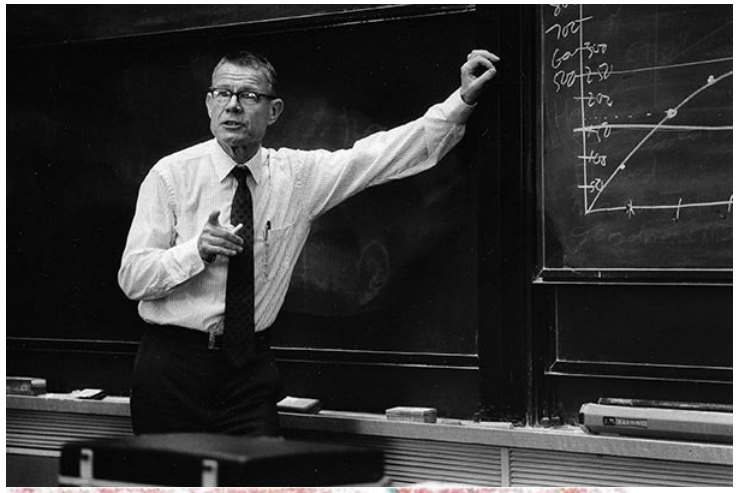
# Shellings Segregation Game

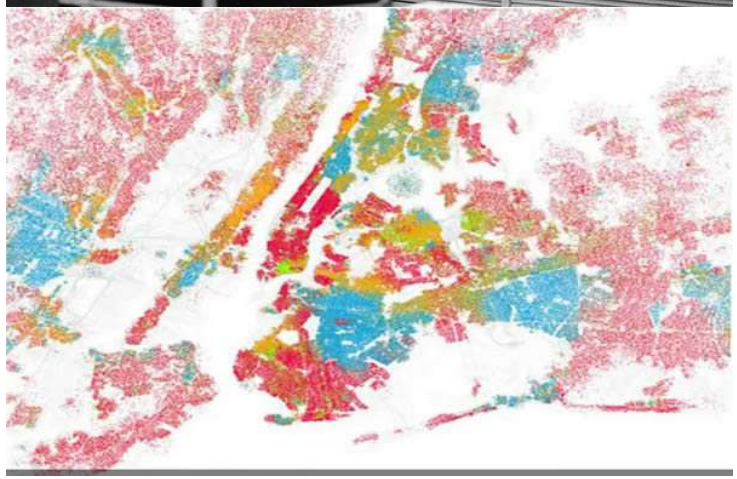
- Very simple model, people move and slightly prefer to be among themselves
- Small bias leads to strong segregation over time
- Chaos: small differences in initial conditions can lead to completely different outcome
- Economically, segregation is optimal, as it maximizes utility. Economists take preferences as given, do not judge. (Maybe they should.)

→Singapore mandates racial mixture of tenants in each block

→San Francisco randomly assigns children to schools across town (potentially leading to large travel times)

Generally: be careful with political (or any other) interpretations. Your results might be an artifact of a programming error, some overlooked detail, or an invalid assumption.





# Break

Forming of teams.

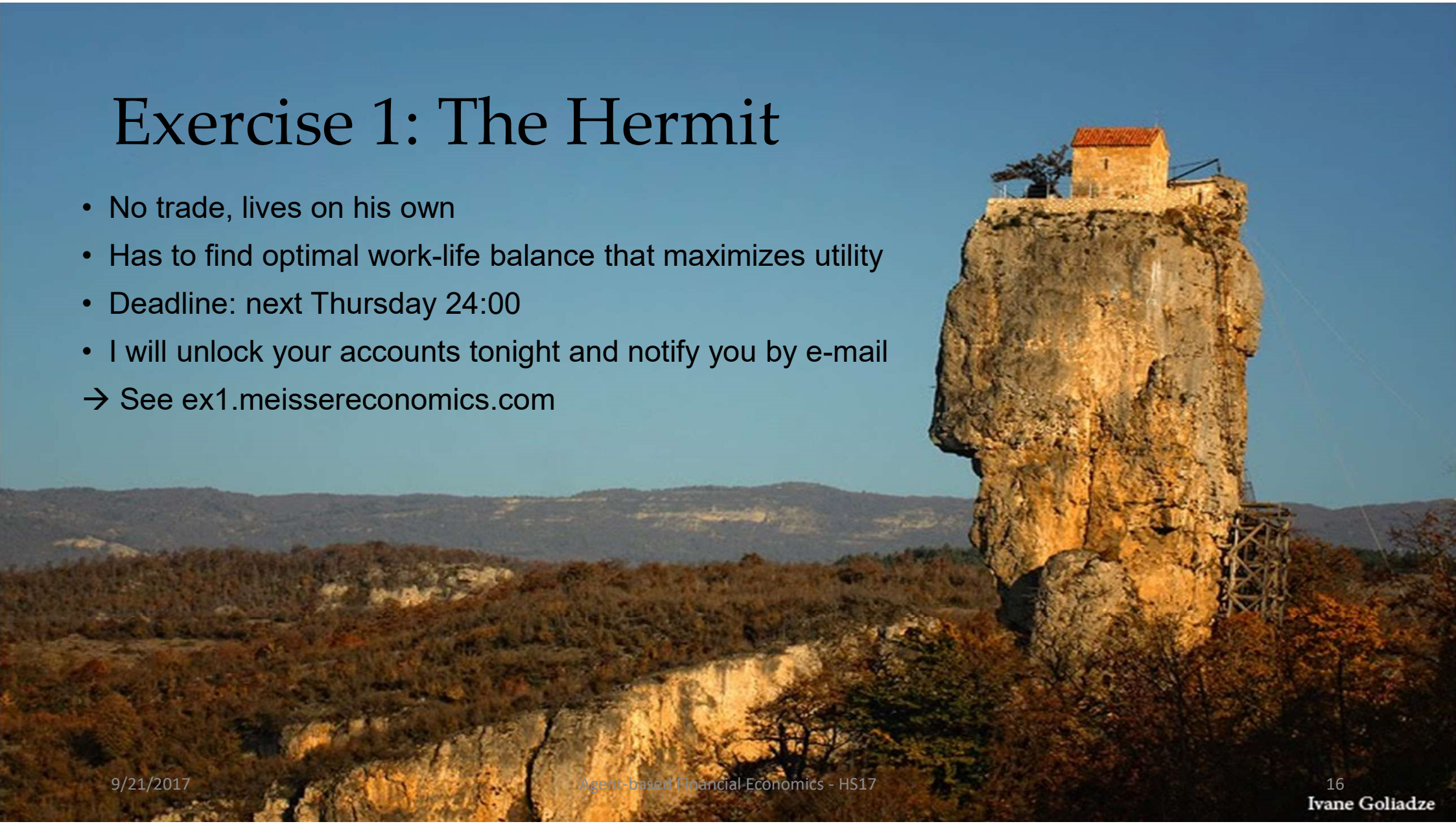
# Our Toy Economy: Basic Setting

- Classic sequence economy with production
- Consumers with utility functions and endowments, trade to maximize discounted lifetime utility
- Price-taking firms with production functions, trade to maximize discounted lifetime profits
- On each day, a pareto-efficient equilibrium must exist (first theorem of welfare economics) and is reachable through trading
- Not clear how to reach it in general (almost “NP hard”)

# Exercise 1: The Hermit

- No trade, lives on his own
- Has to find optimal work-life balance that maximizes utility
- Deadline: next Thursday 24:00
- I will unlock your accounts tonight and notify you by e-mail

→ See ex1.meissereconomics.com



# The Mathematical View

$$U(h_{leisure}, x_{potatoes}) = log(h_{leisure}) + log(x_{potatoes})$$

Thus, the hermit enjoys eating potatoes and spending man-hours as leisure time equally. In order to maximize utility, he needs to decide how much of his 24 hours to spend on leisure time and how much on growing potatoes according to the following budget constraint:

$$h_{leisure} + h_{work} = 24$$

The hours spent working are turned into potatoes via a Cobb-Douglas production function with fixed costs, with $x_{land} = 100$ being constant:

$$x_{potatoes}(x_{land}, h_{work}) = (h_{work} - 6)^{0.6} x_{land}^{0.2}$$

The fixed costs of six hours represent the daily amount of work needed before actual production can start, for example for maintaining the required infrastructure. Plugging the production function and the budget constraint into the utility function, this leads to the following simplified maximization problem:

$$max\ U(h_{work}) = log(24 - h_{work}) + log((h_{work} - 6)^{0.6} x_{land}^{0.2})$$

→ Trivial for the economists among you.

# Many ways to solve this

- Exogenous trial and error
- Calculate optimum and hardcode result
- Calculate analytical solution and calculate optimum dynamically from the current parameters
- Endogenous trial and error
- Golden ratio search
- “Steepest descent”
- Own ideas?

Feel free to do what you feel most comfortable with, but document it!

# Heuristic

What is a heuristic?

A heuristic is a simple recipe to solve a problem well enough.

Example: 1/n heuristic in investing.

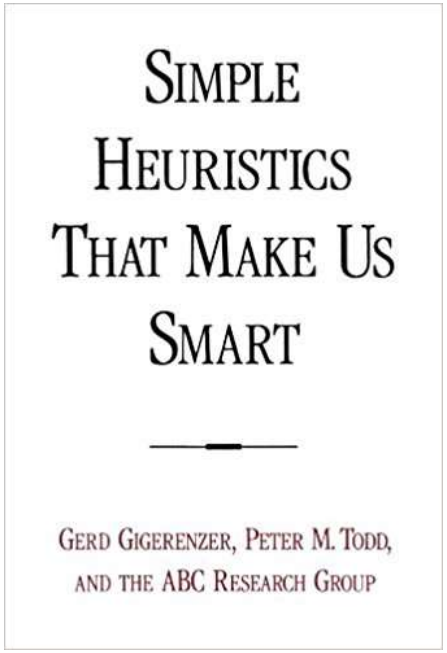
Example: if you don't want to evaluate all the yoghurts at Coop, just buy the same as you bought the last time.

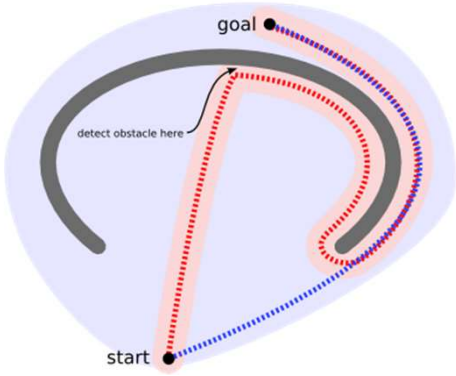
Example: Ant routes.

Related: "greedy algorithms" in computer science.

A good heuristic allows an agent to behave successfully (or even optimally) without spending much resources on solving the underlying problem, at least most of the time.

Gerd Gigerenzer is the "heuristics pope".



# Choice of Tools

- Programming language: Java
- Code editor: Eclipse
- Version control: Git, Github.com, SourceTree
- Documentation: Markdown

Follow the instruction on
http://meissereconomics.com/course/setup
to install all of the above.

# Object-Oriented Programming

- The most popular way of organizing software
- Well-suited to structure complex systems
- Objects encapsulate concerns, hide complexity
- The definition of what an object does is called “class” (cookie form)
- Instances of a class are called “objects” (cookie)
- Classes have two kinds of members: variables and functions (sometimes also called field and method)
- These members might or might not be visible to other classes

→ Particularly well-suited for articulating agent-based models

Class (object definition)

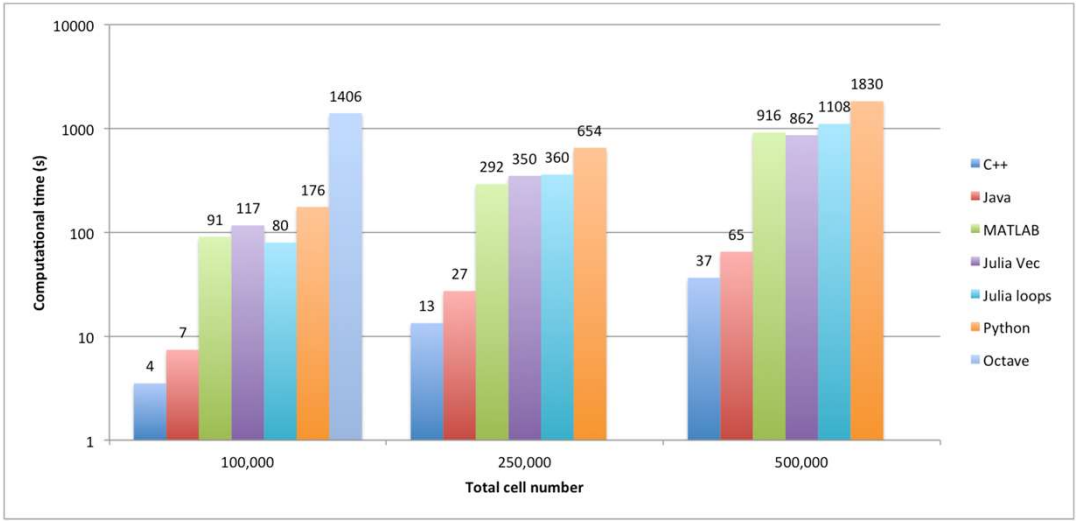
Objects (instances of a class)

# Java

- Most popular programming language by far. Most popular also for agent-based economics.
- Almost as fast as C/C++, but much fewer ways to shoot your own foot.
- Python is also popular among scientists, but about 10 times slower.
- Good to have on your CV!

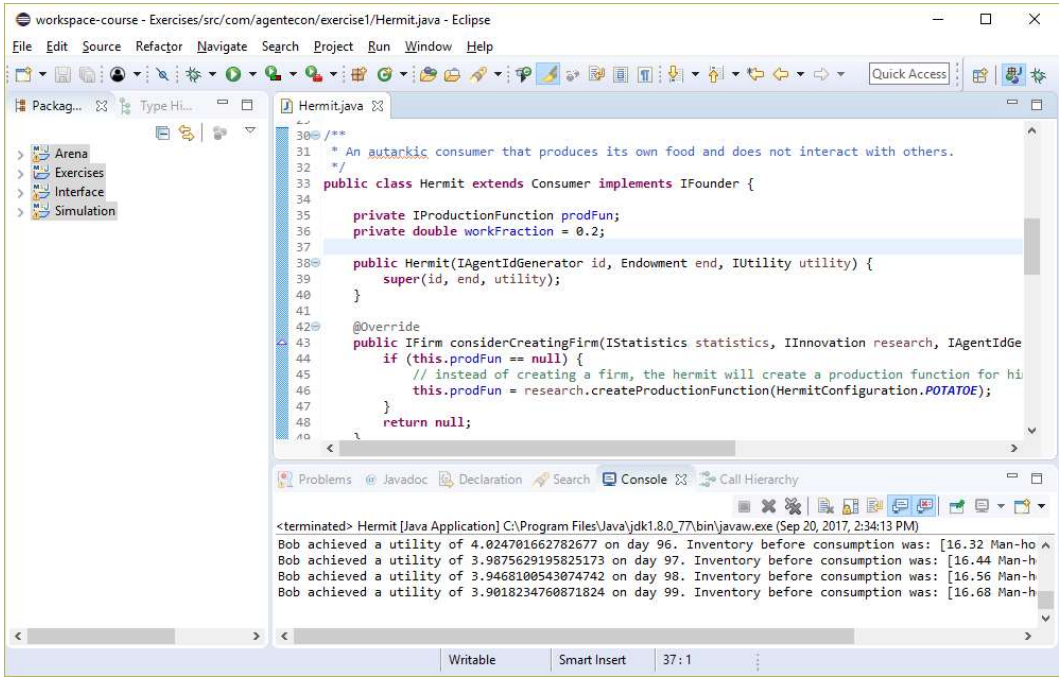
| Sep 2017 | Change | Programming Language | Ratings |
|---|---|---|---|
| 1 | | Java | 12.687% |
| 2 | | C | 7.382% |
| 3 | | C++ | 5.565% |
| 4 | | C# | 4.779% |
| 5 | | Python | 2.983% |
| 6 | ^ | PHP | 2.210% |
| 7 | v | JavaScript | 2.017% |
| 8 | ^ | Visual Basic .NET | 1.982% |
| 9 | ^ | Perl | 1.952% |
| 10 | ^ | Ruby | 1.933% |

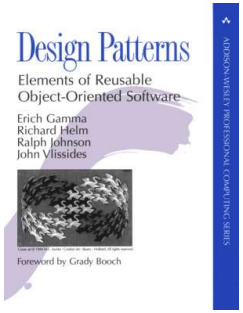
Popularity




Speed

- Created by consortium about IBM as a strategic move against Sun Microsystems – hence the name
- Architect is software engineering heavyweight Erich Gamma who lives in Zurich, currently working for Microsoft.
- Invaluable programming tool, supports the programmer in many way. But not very intuitive at first.




[CITATION] Design patterns: elements of reusable object-oriented software
E Gamma - 1995 - Pearson Education India
Cited by 38430 Related articles All 72 versions Cite Save More

Prospect theory: An analysis of decision under risk
D Kahneman, A Tversky - Econometrica: Journal of the econometric society, 1979 - JSTOR
This paper presents a critique of expected utility theory as a descriptive model of decision making under risk, and develops an alternative model, called prospect theory. Choices among risky prospects exhibit several pervasive effects that are inconsistent with the basic tenets of utility theory. In particular, people underweight outcomes that are merely probable in comparison with outcomes that are obtained with certainty. This tendency, called the ...
Cited by 46464 Related articles All 103 versions Cite Save

```java
/**
 * An autarkic consumer that produces its own food and does not interact with others.
 */
public class Hermit extends Consumer implements IFounder {

    private IProductionFunction prodFun;
    private double workFraction = 0.2;
```

← Comment

← Class declaration. Hermit inherits functionality from the Consumer class. The Hermit also implements the Ifounder interface, allowing him to obtain a production function in our simulation even though he does not found a firm (yet).

There is a field prodFun of type “IProductionFunction” which allows the Hermit to remember his production function. In the beginning, it is empty (null).

There is a private double-precision number that is initially set to 0.2.
Private means that no other class can access this variable.

The class is public, so anyone can use it.

```java
/**
 * An autarkic consumer that produces its own food and does not interact with others.
 */
public class Hermit extends Consumer implements IFounder {

    private IProductionFunction prodFun;
    private double workFraction = 0.2;

    public Hermit(IAgentIdGenerator id, Endowment end, IUtility utility) {
        super(id, end, utility);
    }
```

The constructor is a special function that is invoked whenever a new Hermit is being instantiated. Unlike other functions, it has no return value.

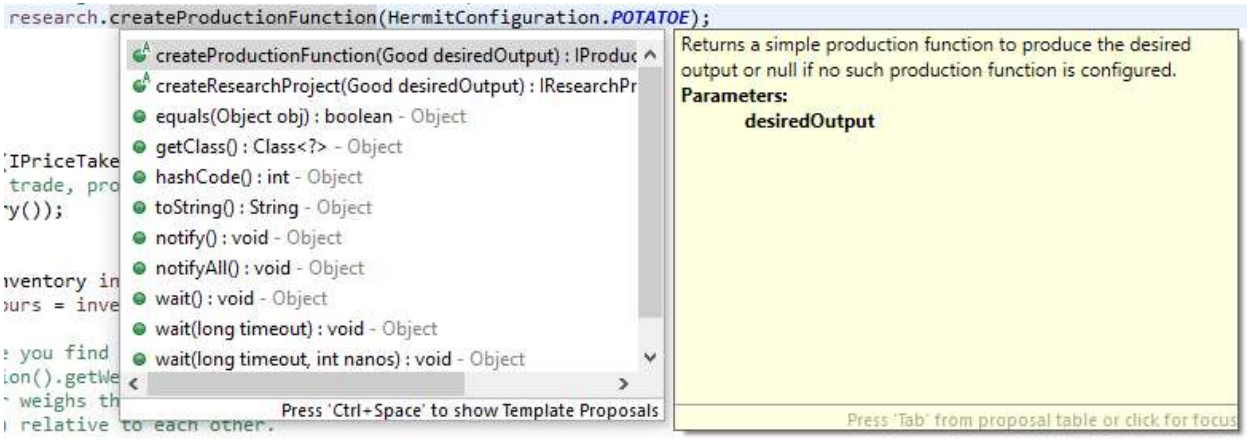
This constructor does nothing except passing on its parameters to the parent (super) class Consumer, which needs them.

Hold Ctrl and click "super" or "Consumer" further above to directly jump to the Consumer class. This is the single most important key shortcut to remember!

```java
/**
 * An autarkic consumer that produces its own food and does not interact with others.
 */
public class Hermit extends Consumer implements IFounder {

    private IProductionFunction prodFun;
    private double workFraction = 0.2;

    public Hermit(IAgentIdGenerator id, Endowment end, IUtility utility) {
        super(id, end, utility);
    }

    @Override
    public IFirm considerCreatingFirm(IStatistics statistics, IInnovation research, IAgentIdGenerator id) {
        if (this.prodFun == null) {
            // instead of creating a firm, the hermit will create a production function for himself
            this.prodFun = research.createProductionFunction(HermitConfiguration.POTATOE);
        }
        return null;
    }
```

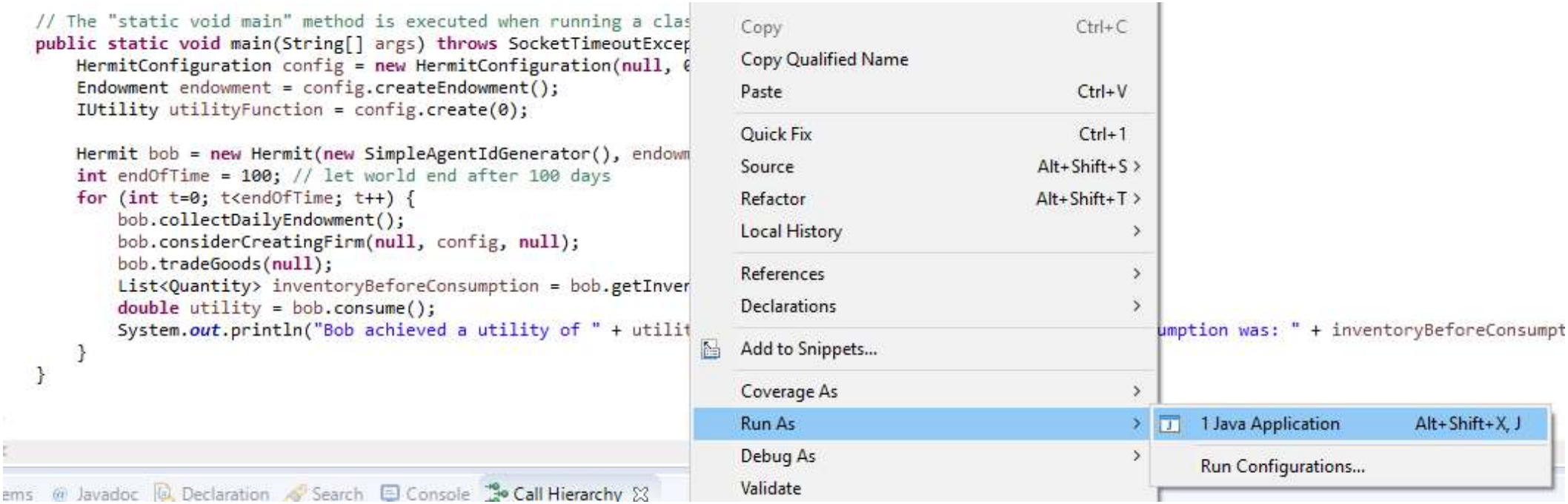
I programmed the simulation to ask all IFounders whether they want to found a firm every day. The Hermit uses this as a hack to obtain access to a productivity function.

If you do not know what functionality an object offers, enter its name followed by a dot and hit "ctrl-space". → Instant documentation!



```java
@Override
public IFirm considerCreatingFirm(IStatistics statistics, IInnovation research, IAgentIdGenerator id) {
    if (this.prodFun == null) {
        // instead of creating a firm, the hermit will create a production function for himself
        this.prodFun = research.createProductionFunction(HermitConfiguration.POTATOE);
    }
    return null;
}

@Override
public void tradeGoods(IPriceTakerMarket market) {
    // Hermit does not trade, produces instead for himself
    produce(getInventory());
}

private void produce(Inventory inventory) {
    IStock currentManhours = inventory.getStock(HermitConfiguration.MAN_HOUR);

    // Play here. Maybe you find a better fraction than 60%?
    // getUtilityFunction().getWeights() might help you finding out
    // how the consumer weighs the utility of potatoes and of leisure
    // time (man-hours) relative to each other.
    double plannedLeisureTime = currentManhours.getAmount() * workFraction;
    workFraction = workFraction + 0.005;

    // The hide function creates allows to hide parts of the inventory from the
    // production function, preserving it for later consumption.
    Inventory productionInventory = inventory.hide(HermitConfiguration.MAN_HOUR, plannedLeisureTime);
    prodFun.produce(productionInventory);
}
```

The simulation invokes the “tradeGoods” function on every consumer every day. However, the Hermit being a hermit does not trade. Instead, he produces his own potatoes feeding his previously obtained production function with some of his 24 man-hours.




Finally, there is a “main” method. Classes that have a main method can be run.
The main method is “static”, meaning that it exists independently of any specific Hermit instance.
To run the “main” method, right-click and select “run as” “Java Application”.

→ Live demo including short tour through the debugger.

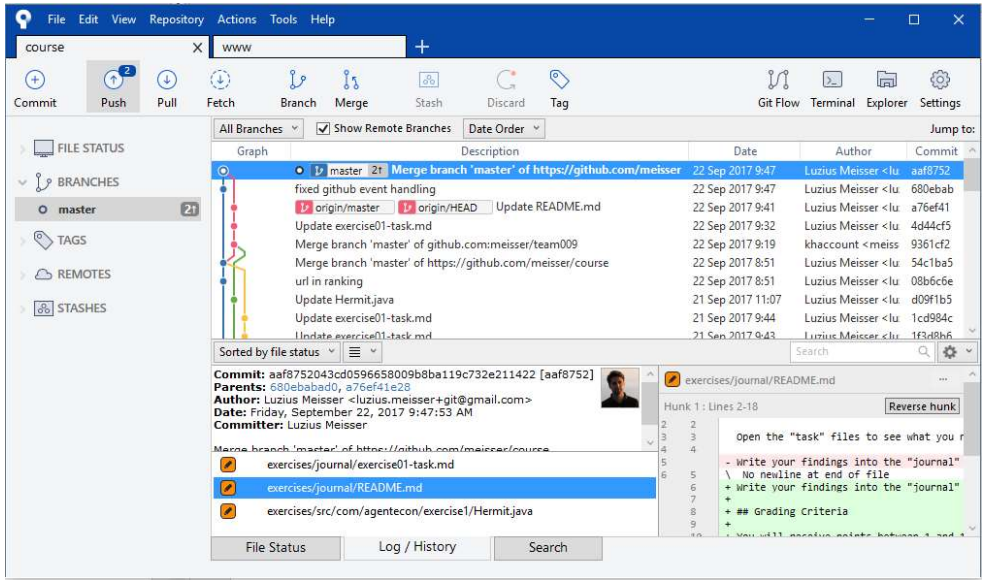
# Version Control

Everything hosted on github.com.
We use the SourceTree client to access it.

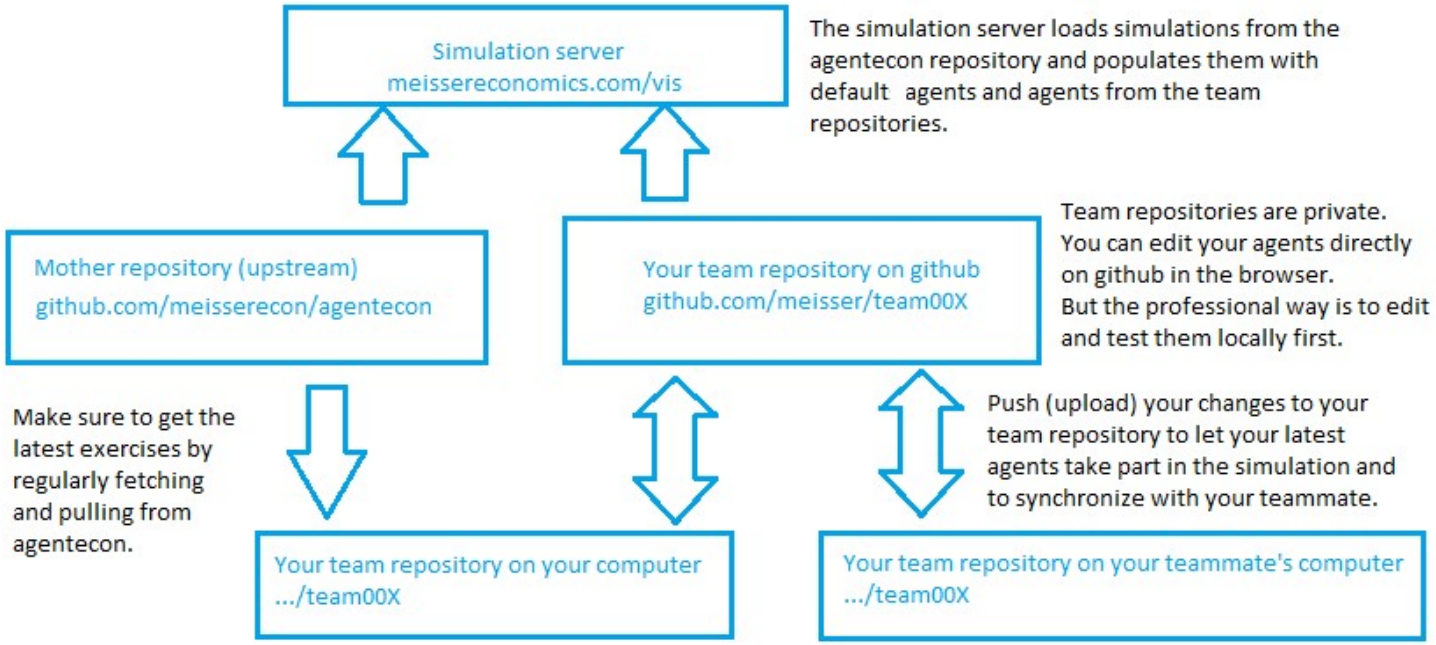
Git is a version control system invented by Linus Thorvalds (the guy who created Linux). Most popular by far, even Microsoft is using Git to store the source code of Windows.

Git stores a history of all you did in a local database located in a subfolder ".git". Nomenclature:

commit: save local changes to the local git database
discard: undo a local change by restoring the previous state from the local git database
fetch: find out what changes are available for download
pull: download changes
push: upload changes
merge: integrate changes from two different sources

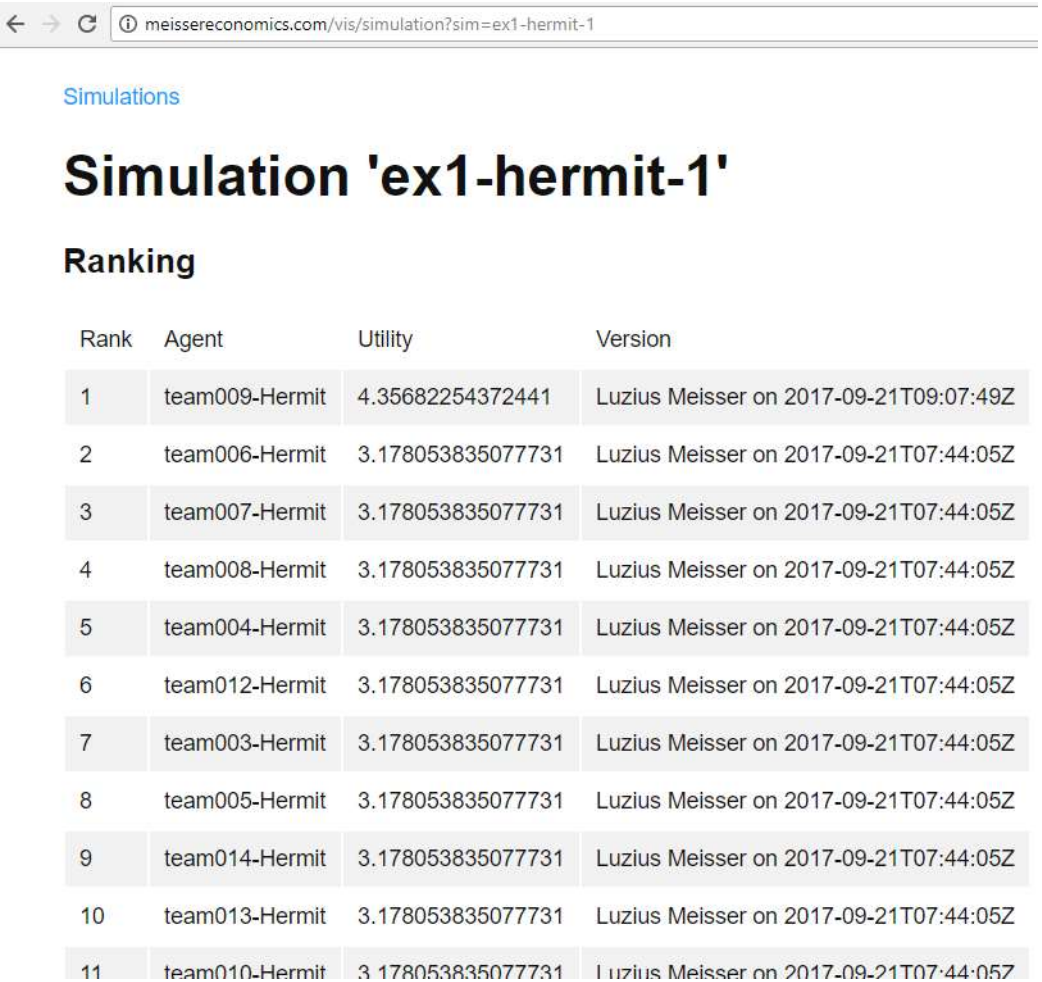


# Version Control




http://meissereconomics.com/course/setup

# Ranking




| Rank | Agent | Utility | Version |
|---|---|---|---|
| 1 | team009-Hermit | 4.35682254372441 | Luzius Meisser on 2017-09-21T09:07:49Z |
| 2 | team006-Hermit | 3.178053835077731 | Luzius Meisser on 2017-09-21T07:44:05Z |
| 3 | team007-Hermit | 3.178053835077731 | Luzius Meisser on 2017-09-21T07:44:05Z |
| 4 | team008-Hermit | 3.178053835077731 | Luzius Meisser on 2017-09-21T07:44:05Z |
| 5 | team004-Hermit | 3.178053835077731 | Luzius Meisser on 2017-09-21T07:44:05Z |
| 6 | team012-Hermit | 3.178053835077731 | Luzius Meisser on 2017-09-21T07:44:05Z |
| 7 | team003-Hermit | 3.178053835077731 | Luzius Meisser on 2017-09-21T07:44:05Z |
| 8 | team005-Hermit | 3.178053835077731 | Luzius Meisser on 2017-09-21T07:44:05Z |
| 9 | team014-Hermit | 3.178053835077731 | Luzius Meisser on 2017-09-21T07:44:05Z |
| 10 | team013-Hermit | 3.178053835077731 | Luzius Meisser on 2017-09-21T07:44:05Z |
| 11 | team010-Hermit | 3.178053835077731 | Luzius Meisser on 2017-09-21T07:44:05Z |

Based on an exponential moving average.

Defined in com.agentecon.web.methods.UtilityRanking
In eclipse, hit Ctrl-Shift-T to quickly find classes by name.

$$s_t = 0.02u_t + 0.98s_{t-1}$$

Note that the following metrics are equivalent in expectation as long as the discount rate corresponds to the probability of death:

- Total life-time utility
- Utility on the last day
- Exponential moving average

But not: average daily utility!!!

Proof in:
https://github.com/meisser/course/blob/master/simulation/documentation/Utility%20Metric.pdf