Course designed as lecture plus tutorials

- 6 ECTS requires investing a **full day, weekly**
- plus additional hours to recap basics
- **weekly assignments** to deepen lecture topics

Unfortunately, AI is a vast and tricky topic. Thus, the course is fast-paced; we expect basic knowledge, in theory and practice

- profound knowledge in **programming** and basic **algorithms**, including complexity analysis
- profound knowledge of basic **math** and basic **stochastics**
Lectures:

- content matched by standard AI text books (recommendations → end of today’s lecture)
- lectures start with quiz questions for self-assessment

Practicals:

- discuss your solutions, no blue-print solutions will be published
- programming tasks in AI programming languages, including Scheme, Prolog, PDDL, and Python
You think you know when you learn, you are more sure when you can write, even more when you can teach, but certain when you can program. (Alan Perlis)

Mode of grading is written exam (Klausur). You must be registered in FlexNow to be allowed to take exams.

- You can earn bonus points by working on (programming) assignments (→ details explained in tutorials)
  - up to 10 bonus points on assignments (due within one week)
  - up to 5 bonus points on a special challenge (due within a month, published in June)

- Bonus points are not required for earning 100% in exam
- **However:** You will not pass the exams without having worked on all assignments
  - must be able to program, must be able to prove
  - binge learning will fail, study regularly
Planned Schedule

14 weeks of semester, but 2 lectures and 2 tutorials “co-located” with public holidays → two separate video-lectures

<table>
<thead>
<tr>
<th>date</th>
<th>lecturer</th>
<th>topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>04/25</td>
<td>DW</td>
<td>organisation, introduction to AI</td>
</tr>
<tr>
<td>05/02</td>
<td>DW</td>
<td>search foundations</td>
</tr>
<tr>
<td>05/09</td>
<td>DW</td>
<td>heuristic search</td>
</tr>
<tr>
<td>05/16</td>
<td>DW</td>
<td>search in game-playing</td>
</tr>
<tr>
<td>05/23</td>
<td>DW</td>
<td>knowledge representation foundations</td>
</tr>
<tr>
<td>05/30</td>
<td>DW</td>
<td>(holidays) video: classic logics</td>
</tr>
<tr>
<td>06/06</td>
<td>US</td>
<td>resolution and logic programming</td>
</tr>
<tr>
<td>06/13</td>
<td>US</td>
<td>non-classic logics</td>
</tr>
<tr>
<td>06/20</td>
<td>US</td>
<td>(holidays) video: planning</td>
</tr>
<tr>
<td>06/27</td>
<td>US</td>
<td>machine learning</td>
</tr>
<tr>
<td>07/04</td>
<td>DW</td>
<td>object recognition</td>
</tr>
<tr>
<td>07/11</td>
<td>DW</td>
<td>natural language processing</td>
</tr>
<tr>
<td>07/18</td>
<td>US</td>
<td>cognitive and situated AI</td>
</tr>
<tr>
<td>07/25</td>
<td>US</td>
<td>AI applications</td>
</tr>
</tbody>
</table>
What is Artificial Intelligence (AI)?

Artificial intelligence is the science of making machines do things that would require intelligence if done by men. (Minsky, 1963 – co-founder of AI discipline)

Defining AI is challenged by problematic term “intelligence”

- operational definition in psychology: 
  *intelligence is what is measured by an intelligence test*

- analytical definition: 
  *intelligence is the ability to acquire and apply knowledge*

- even systems of very limited intelligence can be useful 
  *e.g., a speech recognition vs. understanding language*

Even worse:

- some problems where we ascribe high intelligence to a human capable of solving are easy for computing machinery (e.g., solving mathematical equations)

- others are easily solvable by humans, but still hard/unsolved for artefacts (e.g., communicating in natural language, pouring liquids, recognising close friends, . . .)
We may consider AI as the discipline striving to understand intelligent behaviours and discovering means to design such artefacts. It is best characterised by research aims it encompasses:

- creating artefacts that act intelligently
- devising methodologies to implement these systems
- grasping natural intelligence from an information-processing point of view
(human intelligence makes heavily and easy use of context, Selfridge, 1955)
Example: Context Effects

(human intelligence makes heavily and easy use of context, Selfridge, 1955)
The extent to which we regard something as behaving in an intelligent manner is determined as much by our own state of mind and training as by the properties of the object under consideration. If we are able to explain and predict its behavior we have little temptation to imagine intelligence. With the same object, therefore, it is possible that one man would consider it as intelligent and another would not; the second man would have found out the rules of its behavior.

(Alan Turing, 1947)
Further Definitions of AI

There is no really satisfying definition of AI, two which I like are:

*AI researches how one could make a computer do what humans currently can do better.* (Elaine Rich)

*AI research is concerned with computer problems yet unsolved.* (Marvin Minsky)

- For many concepts/ scientific fields (not only for AI) there is no single satisfying definition.
- Pragmatic solution: Enumerate the topics, give examples (getting “concrete”)*
• AI is an integral part of computer science, the beginnings of computer science research are also the beginnings of AI research, but:

• \textit{AI is that sub-discipline of computer science, which is concerned with analysis and formalization of thought processes which are not yet formally understood. In the case of success, that is if these processes got “de-mystified”, these areas become part of standard computer science and AI research is moving on to new problems.} (Christian Freksa)

Examples:

- problem solving $\rightarrow$ efficient search algorithms
- automatic programming $\rightarrow$ compiler design
- reasoning $\rightarrow$ theorem proving
AI as formal science

- **Analytical, formal:**
  Analysis of problems, formalization, design and evaluation of algorithms (completeness, correctness, optimality; empirical evaluation of error rates and performance times)

  \[ \rightarrow \text{Precise description of problems and algorithms; formal, normative constraints for performance} \]
AI as natural/epistemological science

- **Empirical:**
  Recourse to biological and cognitive principles of organization and processing and their simulation
  ↪ *Evidence for the existence of abilities and skills, their constraints and characteristics of performance*
AI as engineering science

- **Engineering:**
  (Efficient) implementations of algorithms, adaptation of algorithms to requirements of specific applications
  \(\Rightarrow\) *Application is one possibility to evaluate the scope, functionality and relevance of AI-algorithms*

\(\Rightarrow\) As an AI researcher you should have knowledge of formal methods and of cognitive theories and empirical methods!
Two Perspectives

- AI as engineering discipline:
  make algorithms for solving AI problems

- AI as cognitive modeling:
  make algorithms which solve problems in a similar fashion as humans
  \(\leftrightarrow\) **AI as theoretical psychology** (cognitive AI)

*Everything is vague to a degree you do not realise till you have tried to make it precise.* (Bertrand Russell)
• **Problem solving and planning**
  - basic search techniques for many AI topics
  - Applications:
    Scheduling, configuration, manipulation of formula, games

• **Inference and theorem proving**
  - deduction of information which is given implicitly in knowledge bases
  - in contrast to other areas of AI:
    algorithms which are provably complete and correct
  - non-monotonic reasoning, fuzzy/probabilistic approaches
  - Applications:
    Program verification, knowledge based/expert systems
• **Knowledge representation**
  - can be crucial for performance of AI systems (easy retrieval of stored knowledge)
  - logic, semantic nets, frames/schemes, graphs, ...

• **Machine learning**
  - Concept learning, classification learning
  - Strategy learning, policy learning, inductive programming
  - Applications: data mining, object recognition, process control
Areas of Application

- Natural language processing
- Computer vision (object recognition)
- Multi-Agent-Systems
- Robotics (action planning, navigation)
- ...

AI programming

- Declarative programming languages (5th generation languages): Prolog, Lisp; today there are no longer typical AI languages. AI tools for industrial applications are written in C++, yet researchers employ languages supporting rapid prototyping.
Relations to Others Disciplines

- **Philosophy:**
  Early AI made strong promises, philosophers started to ask whether human cognition has inherent aspects which cannot be transferred to a computer (Dreyfus, Searle). AI is an interesting area for philosophical studies of mind.

- **Linguistics:**
  Chomsky had very strong influence to computer science (Chomsky hierarchy) and had a major impact on the beginning of cognitive psychology (Chomsky-Skinner debate).

Interaction of AI and Linguistics:
Computer Linguistic

- **Psychology:** AI models influence psychological models (Minsky’s computer metaphor Mind/Brain, Software/Hardware).
- Kasparov vs. Deep Blue, Match 1996: Kasparov suspected human intervention
- Searle critique: Thought experiment “Chinese Room”
- Cognitive Science approach: do not judge superficial performance; investigate process characteristics of the program (correspondence of number of rule applications to reaction times, similar errors etc.)
A Short History of AI

Prehistory

1943 McCulloch & Pitts “Logical Calculus of the Ideas Immanent in Nervous Activity” (Architecture for an intelligent system based on a neural net)

1948 Wiener “Cybernetics” (information theory)

1950 Turing: “Computing Machinery and Intelligence” (Turing-Test)

1955 Selfridge:
Pattern-matching program

(from Lindsay and Norman, 1977)
A Short History of AI cont.

Early AI (1956 to mid 60ies)

- Dartmouth conference of 1956: Term AI comes into use (McCarthy)
- Focus on models of cognitive processes and general principles of intelligent behavior
  - Marvin Minsky (MIT): Perception
  - A. Newell and H. Simon (CMU): Inference - theorem prover for propositional logic “Logic Theorist” (1956)
    Problem Solving: GPS (1958)
  - John McCarthy (Stanford): Lisp (1958)
... **Early AI** (1956 to mid 60ies)

- Games: Samuels (1959), Checkers
- Learning: Winston (1970), Learning by analogy
- Analogical reasoning: Evans (1959)
- Simon, 1965: “*by 1985 machines will be capable of doing any work a man can do*”
- Minsky (1968) “Semantic Information Processing” (an important collection of early work)
- Drawbacks, e.g. in machine translation $\rightarrow$ AI critique
Evans (1968), grammar inference method

A

B

C

1

2

3

4

5
The middle years (70ies)

- no interest in interdisciplinarity and cognition
- focus on knowledge: representation and inferences
- Question-Answering systems, Expert systems
- SHRDLU, Winograd (1970): natural language system for manipulation and verification of statements in a blocksworld (procedural semantics)
- DENDRAL, Feigenbaum (since 1965): Analysis of molecular structures
- MACSYMA (1971): manipulation of formulas
- MYCIN, Shortliff (since 1974): medical diagnosis
... The middle years (70ies)

- PROLOG (1973) Colmerauer
- Development of languages for production systems (OPS, McDermott 1977) and shells for expert systems
- Planning: STRIPS (Fikes & Nilsson 1971), NOAH (Sacerdoti 1975)
- non-classic logic (Reiter, 1980), fuzzy logic (Zadeh, 1965)
- hierarchical nets, Quinlan (TLC, 1968); frames, Minsky (1975); Scripts, Schank & Abelson (1973)
The 80ies

- Search for real world applications; intelligent tutor systems
- Language research: functional-logic programming
- Re-invention of backpropagation, new interest in neural/distributed information processing (Feldmann & Ballard 1982; Rumelhart & McClelland 1986)
- Big battles of symbolic vs. sub-symbolic AI
The 90ies and now

• new interest in interdisciplinary research, birth of cognitive science

• AI-methods are used when helpful:
  “Some of the most successful applications of AI are those in which the artificial intelligence is spread like raisins in a loaf of raisin bread: the raisins do not occupy much space, but they often provide the principal source of nutrition.”
  (Esther Dyson, industrial analyst)

• Focus on learning

• Relation to robotics: embodied intelligence, situated action

• Multi-Agent Systems

Armar, the humanoid robot (KIT), see video
community.aldebaran-robotics.com/
Many AI programming techniques became mainstream

- OOP, lambda abstraction, pattern matching, expression manipulation, evaluation, ...
- still complicated to use with some environments (e.g., evaluation, pattern matching in Java)

For several reasons it is thus helpful to study classic AI languages

- ease of use of concepts
- support for rapid prototyping and experimentation

Classic AI programming was designed for programmers who did not know how to solve the programming task when starting to program – experiments with coding to grasp the problem.
In this course we will meet *interactive languages* that support rapid prototyping and exhibit unique programming features. **Read-Eval-Print-Loop (REPL)** implements the idea that program execution consists out of four phases:

- **read** reading input, program and data – transform into internal representation, possibly compile input to machine language
- **eval** evaluate expression according to semantics
- **print** print out the result
- **loop** ...and start over with the read step

Every of these phases is implemented in the programming environment and can be changed by the programmer! Almost all scripting languages follow the REPL paradigm, e.g., Python.
What features are important for implementing AI techniques?

- problem-dependent: design programming language for user-defined extensions
- support for handling data other than simple data types (numbers, strings): terms, expressions, program code (sic!)
- minimal core: keep language simple, complexity from extensions

**Example:** Lisp (list processor) treats all data as (dynamic) lists, allows user to change the parser, evaluator, and printer.
Program languages can become messy as they grow (legacy constructs, mix of concepts). Example: partial type inference in Java functional constructs.

Incomplete language influence chart, highlighting languages used in this course.
Scheme has been designed as an educational version of Lisp: Scheme is not relevant for software development, but for learning programming concepts and becoming a better programmer. In this course, we’ll be using the Scheme implementation Racket, which comes with a GUI and graphics support for many operating systems.

Scheme implements REPL and uses a very simple syntax:

- strings, numbers as known from C, Java; plus rationals ($7/3 = \frac{7}{3}$)
- words starting with a non-numeric character and containing letters, digits, and other characters are called symbols: $+$, $=$, foo just-a-name, fahrenheit->celsius, ...
• Input tokens are **separated by space**: foo bar are thus two tokens

• Semicolon `;` marks begin of **comment**, comments end at end of line

• Parentheses `()`, `[,]` group tokens to lists, example: `(+ 2 3)` is the list containing symbol `+`, numbers 2 and 3

• `#t`, `#f` represent the Boolean values true and false

• Prepending **quote** character `'`, `' to an expression is equivalent to writing `(quote ...)`: `'(a b) = (quote (a b))`

• Prepending `,` to an expression is called **backquoting**

Note: `#` as in `#t` marks an escape sequence to the reader in Lisp languages, allowing the language parser to be customised easily. One could, for example, build extensions that allow latin numbers to be input (`#L–MCMLXXIV ↝ 1974`), or complex data types.
Extending syntax can be helpful to keep code compact and improve readability like with a custom `PersonalRecord` reader `#PR`:

```java
PersonalRecord P = new PersonalRecord("John Doe", ...);
PersonalRecord Q = new PersonalRecord("Jane Doe", ...);
List<PersonalRecord> L = new List(P);
L.add(Q);

// (…)

someObject.someMethod(L);
```

(someFunction '(#PR("John Doe" ...) #PR("Jane Doe" ...))

Note that this would not be possible in Java since Java does not follow REPL paradigm: you cannot use your code during compile time!
Although Scheme is special in several details, semantics appear rather straightforward.

- Expressions represent values determined according to an **evaluation rule**
- All objects created by computation have unlimited extent (implementations may reclaim memory that can be proven not affect future computations)
- Scheme has **latent types** which are associated to values, not to variables. I.e., there are no variables of type integer, but only integer values.
- Mathematical approach to numbers: In Scheme, every integer is a rational number, every rational is a real, and every real is a complex number. Integers can be of arbitrary length.
- Procedures are just objects; the state of computation (continuation) is also just an object.

Scheme Semantics Overview
Key to semantics are **evaluation rule** and the semantics of special expressions

- every non-list object which is not a symbol, called **atom**, evaluates to itself: `foo"` ⇝ `foo"`, `4` ⇝ `4`, ...
- symbols evaluate to the value bound to the symbol. It is an error to evaluate an unbound symbol: `+` ⇝ `<procedure:+>`
- lists are evaluated by...
  1. evaluating all their elements
  2. assuming the first element of the list to designate a procedure, call that procedure and passing remaining elements as parameters (call-by-value)
  3. it is an error, if the first element of a list does not evaluate to a procedure

Example: `( + 1 2 )` ⇝ `3`
(define symbol value) evaluates value and then establishes a binding of symbol to that value, example:

```scheme
> (define my-value 42)
> my-value
42
```

To define functions, instead of writing
```
(define add2 (lambda (x y) (+ x y)))
```
one can shortly write
```
(define (add2 x y)
  (+ x y))
```

Indention no formal requirement, but required for readability. Note, there are no closing parentheses in single lines – code shall not look like C/Java.

More special expressions and built-in functions can be found in the Scheme documentation https://docs.racket-lang.org/.
Useful expressions

<table>
<thead>
<tr>
<th>cond</th>
<th>multiple-case if</th>
</tr>
</thead>
<tbody>
<tr>
<td>when, unless</td>
<td>like if, but without else- rsp. then-part</td>
</tr>
<tr>
<td>eqv?</td>
<td>test equivalence of objects</td>
</tr>
<tr>
<td>equal?</td>
<td>test equivalence of structures (recursive eqv?)</td>
</tr>
<tr>
<td>=</td>
<td>test numeric equivalence</td>
</tr>
<tr>
<td>first</td>
<td>first element of list (second, etc. also defined)</td>
</tr>
<tr>
<td>rest</td>
<td>list with first element removed</td>
</tr>
<tr>
<td>cons, append</td>
<td>constructing lists</td>
</tr>
<tr>
<td>let, letrec</td>
<td>locally introducing variables</td>
</tr>
<tr>
<td>set!</td>
<td>update value of variable</td>
</tr>
</tbody>
</table>

**Note:** all test predicates end with ? (eqv?, odd?, even?, ...), all procedures which alter a symbol-value binding end with ! – maintain convention in your code!
*The mostly used text book in the field and main resource for this course*

Peter Norvig (1992). Paradigms of Artificial Intelligence Programming  
*AI programming with Lisp, available online https://github.com/norvig/paip-lisp*

*A not very formal book. It's advantage is that it addresses relations to cognitive science research.*


*Nilsson having invented the A*-algorithm, his chapters on heuristic search algorithms are written very clearly and detailed. His most comprehensive text on search methods is:*


*A detailed overview of topics of AI. Gives an overview of AI-research in Germany.*


*This is not a scientific text book but a most motivating, comprehensive book for a broad readership.*

... many others, see course web page
AI Organizations

- **AAAI**: Association for the Advancement of Artificial Intelligence (AAAI) (formerly the American Association for Artificial Intelligence)
  - Most important international conference: IJCAI
  - Most important journals: Artificial Intelligence (JAIR)
- **ECCAI**: European Coordinating Committee on AI
- **Fachbereich KI, der Gesellschaft für Informatik (GI)**
  - Annual conference: KI, interdisciplinary spring school: IK (Interdisziplinäres Kolleg)
• **Definition(s) of AI:** developing systems for problems which currently are solved better by humans

• **Methods of AI:**
  analytical, empirical, engineering

• **Topics of AI:**
  Knowledge Representation, Inference, Learning, Planning, ...

• **AI Languages**

• **Application areas:**
  Information Systems, Computer Vision, Robotics, ...

• **Relations to philosophy, linguistics, psychology**

• **Turing Test**

• **Early AI researchers:**
  McCarthy (programming languages, logic), Minsky (applications, engineering), Newell & Simon (cognitive AI)