CS 5100: Foundations of Artificial Intelligence

Introduction to AI, Agents, and Python

Prof. Amy Sliva

September 8, 2011

Outline

- What is AI?
- Syllabus and course administration
- (Very) brief history of AI
- Intelligent Agents
- Introduction to Python

What is artificial intelligence?

- Artificial systems with humanlike ability to **think, understand, and reason** (cf. cognitive science)
- Solve problems too large to find the best answer algorithmically
  - **Heuristic** (incomplete) methods
- Solve problems that are **not well-understood**

How do we determine success?

1. Getting the "right answer"?
2. The Turing Test (or modified versions)?
3. Usefulness of the resulting techniques?
4. Know it when we see it?

Artificial systems with humanlike ability to understand and reason

- Main techniques
  - Ontologies, automated reasoning, formal logic, state-space search, evidential logics (probability, fuzzy logic, …), Bayesian inference nets, Markov models

- Applications
  - Problem-solving/planning, natural language processing, intelligence HCI, problem-solving under uncertainty, decision support systems (“expert systems”)

Solve problems too large to find the best answer algorithmically

- Main techniques
  - Heuristic search, dependency-directed backtracking

- Applications
  - Production scheduling and other constraint satisfaction problems, game playing
  - A component in large-scale reasoning and planning systems

Solve problems that are not well-understood

- Main techniques
  - Weighted rule-based systems, Bayesian inference nets, statistical induction and machine learning in general

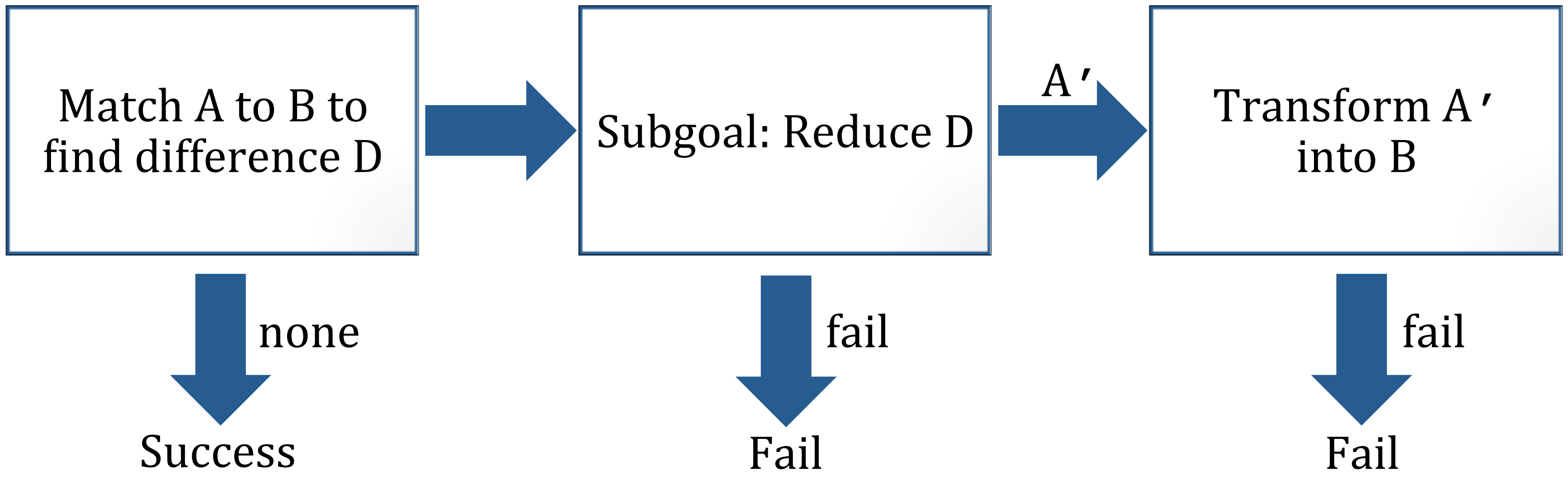
- Applications
  - Finance, search engines, computational science (discovery), data mining
  - Computer vision: systems that see and recognize objects

Go over syllabus

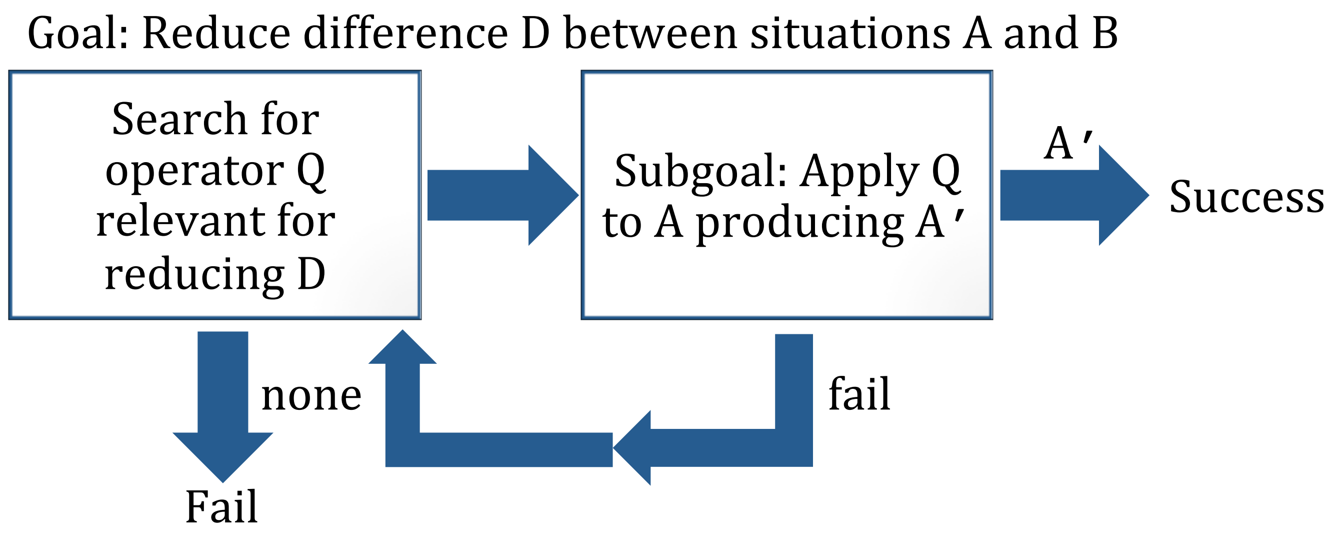
History of AI

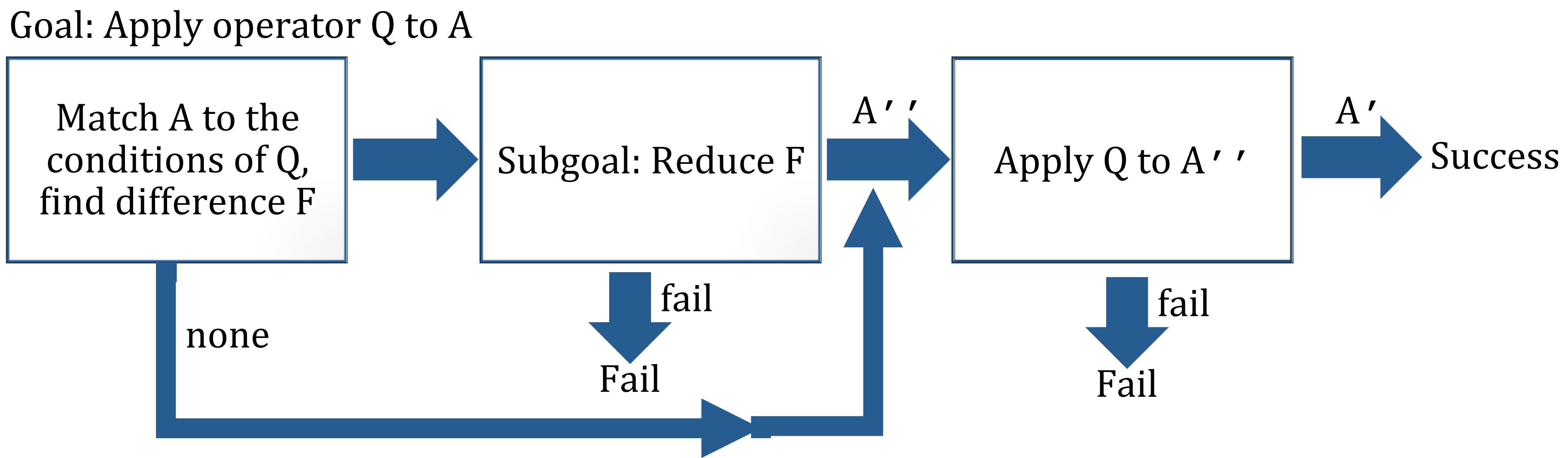
- 1960s—Initial optimism
  - Early ML
  - Samuel’s Checkers player
  - General Problem Solver (GPS)—Simon & Newell
    - Employed **means-ends analysis** (precursor of backward chaining now used in many systems)

Goal: Transform situation A to situation B



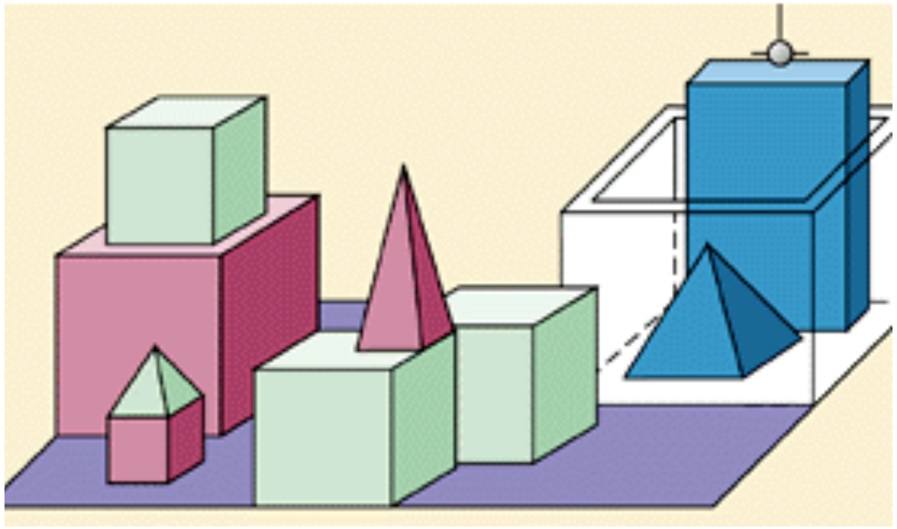
More on means-ends analysis


Goal: Reduce difference D between situations A and B
Search for operator Q relevant for reducing D
Subgoal: Apply Q to A producing A′
A′
Success
none
Fail
fail


Goal: Apply operator Q to A
Match A to the conditions of Q, find difference F
Subgoal: Reduce F
A′′
Apply Q to A′′
A′
Success
none
fail
Fail
fail
Fail

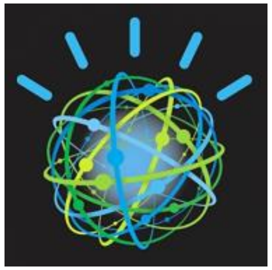
History of AI (cont.)

- 1970s - mid 80s—Knowledge-based systems
  - "Micro-world" experiments
    - SHRDLU (Terry Winograd)
  - Rule-based **"expert" systems**
    - DENDRAL, MYCIN (Ed Feigenbaum)
  - Acceptance by industry—huge oversell
    - The **knowledge acquisition bottleneck**

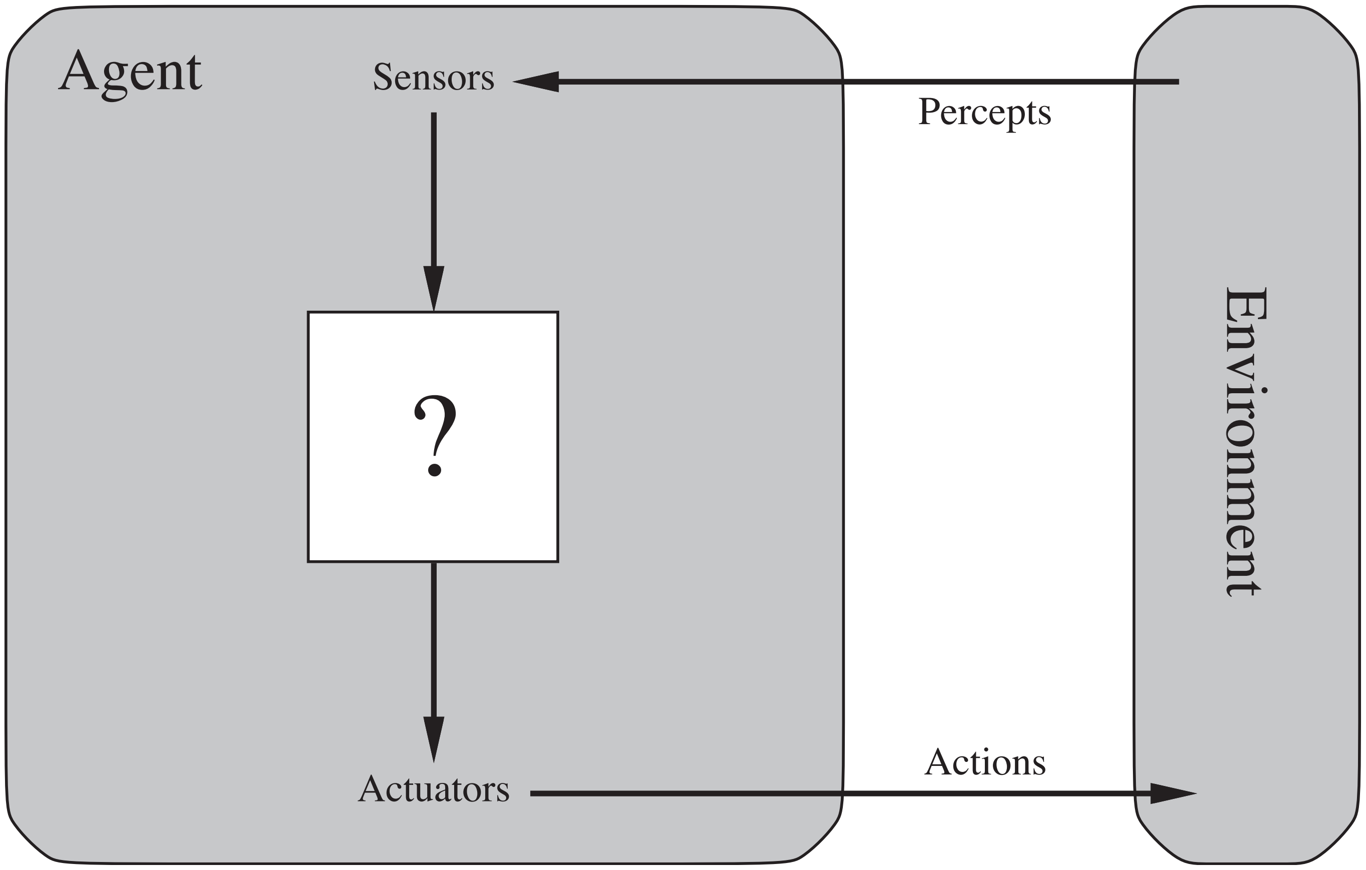


History of AI (cont.)

- Late 80s – mid 90s—AI Winter
  - Hopes pinned on neural nets/ML to overcome KA bottleneck
- Late 90s to present—more computing power
  - Rise of **probabilistic** approaches
    - Lexical tagging breakthrough in NLP
  - More **rigorous** experiments/evaluation methods
- 2000s—influence of the web revives AI
  - Massive text corpuses and need for better web browsers inspire NLP
  - Hardware advances inspire robotics
  - **Intelligent agents**/web bots—applications to e-commerce



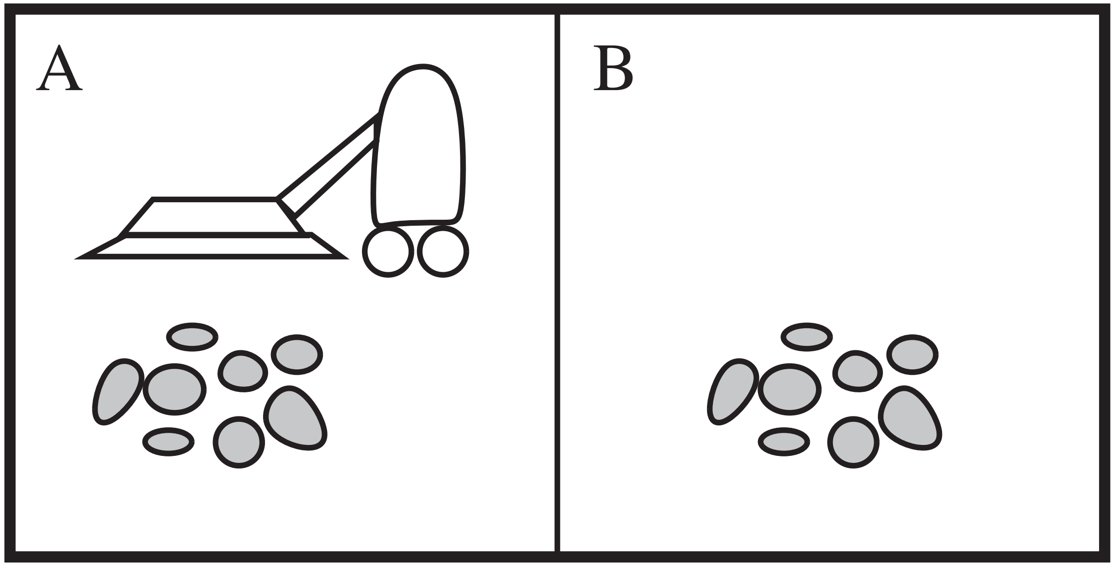
Agents and environments



Framework for intelligent agent design

- What can the agent do? (range of possible **actions**)
- What about the **environment**?
  - Inputs to the program are called **percepts**
    - Symbolic input from keyboard, files, networks
    - Sensor data from the physical world
    - Often must be **interpreted** into meaningful concepts
- What can the agent know?
  - **History** of its own previous inputs and actions
  - Properties of the environment and **world knowledge**
  - Knowledge of its own **goals**, preferences, etc.
  - **Strategies** for its behavior
- Describe the agent's behavior with an **agent function**
  - Mapping of any percept sequence to an action
  - Implemented internally by the agent **program**

Vacuum-cleaner world



- Percepts: location and contents, e.g., [A,Dirty]
- Actions: *Left*, *Right*, *Suck*, *NoOp*

Types of agents

- Simple reflex agent
  - No "**state**" or memory
  - Reacts to current input according to its program (**rules** of the form "**if** *condition* **then** *action*")
- Model-based agent
  - Uses an explicit **knowledge base** to model the environment
    - How does the environment **evolve** independently
    - How does the agent **affect** the environment
  - Exhibits "understanding" of its input by relating it to prior knowledge
  - Reacts according to rules
    - Conditions may be complex and require inference to evaluate

Types of agents (cont.)

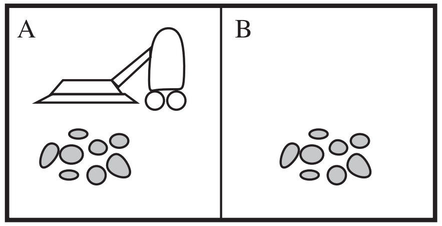
- Planning agents (goal-based and utility-based agents)
  - Explicitly represent their own **goals** and/or **preferences** ("utilities") and can reason about them (i.e., planning)
  - Exhibit autonomy—actions do not follow directly from rule-based lookup
- Learning agents
  - Supervised learning—Learn from positive and negative **examples**
  - Reinforcement learning—Learn from **experience** to improve its outcomes

Agent program implementation

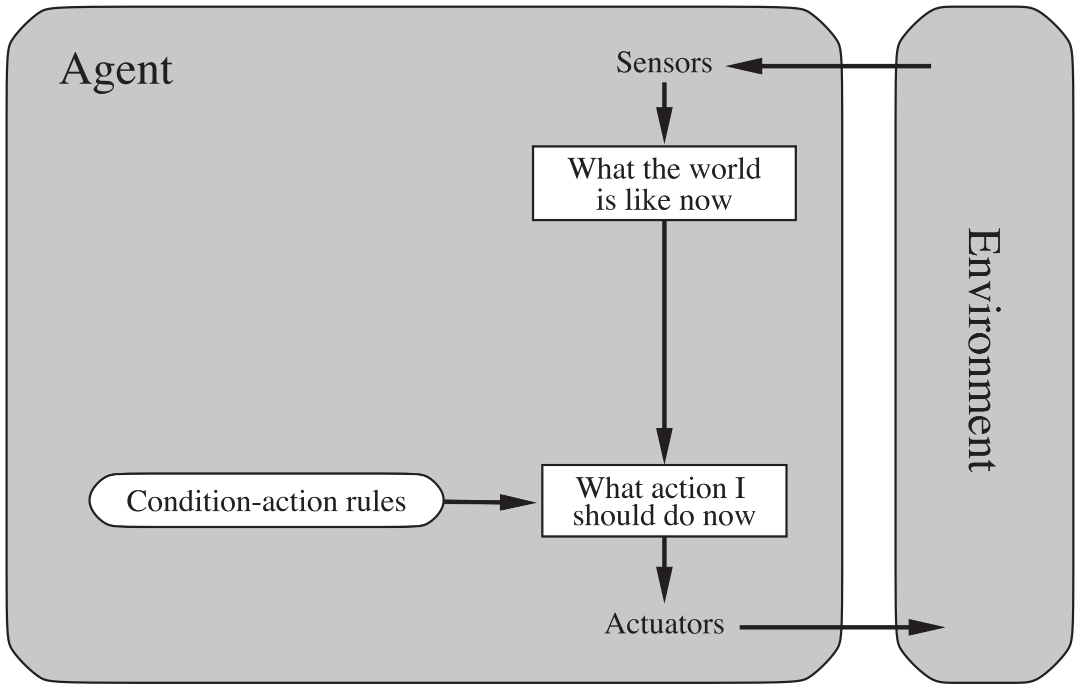
- Table-driven approach—intractable
  - Use lookup table to match the sequence of percepts to an action
- Embedded representation—specific to one environment
  - Program statements:

```
1. if status = Dirty then return Suck
2. else if location = A then return Right
3. else if location = B then return Left
```

- Declarative representation—general
  - Program statements:
    - Use **production rule** base: ***condition*** $\rightarrow$ ***action***
    - If percept matches ***condition*** then return ***action***



Declarative simple reflex agent



- Drawbacks of production rule systems
  - HUGE rule base—time consuming to build by hand
  - What if more than one condition is satisfied?
  - Inflexible (no adaptation or learning)

Representing agent knowledge

- **Q:** What formal language(s) can we use to represent
  1. Current facts about the state of the world?
  2. A model of how the world behaves?
  3. A model of the effects of actions that the agent can perform?
  4. The production rules that specify agent behavior?

- **A:** Formal logic
  - Syntax and semantics are well understood
  - Computational tractability known for important subsets (e.g., Horn clause logic)

How do we determine success?

1. Getting the “right answer”?

2. The Turing Test (or modified versions)?

3. Having a good outcome?
   (using some “utility” function)

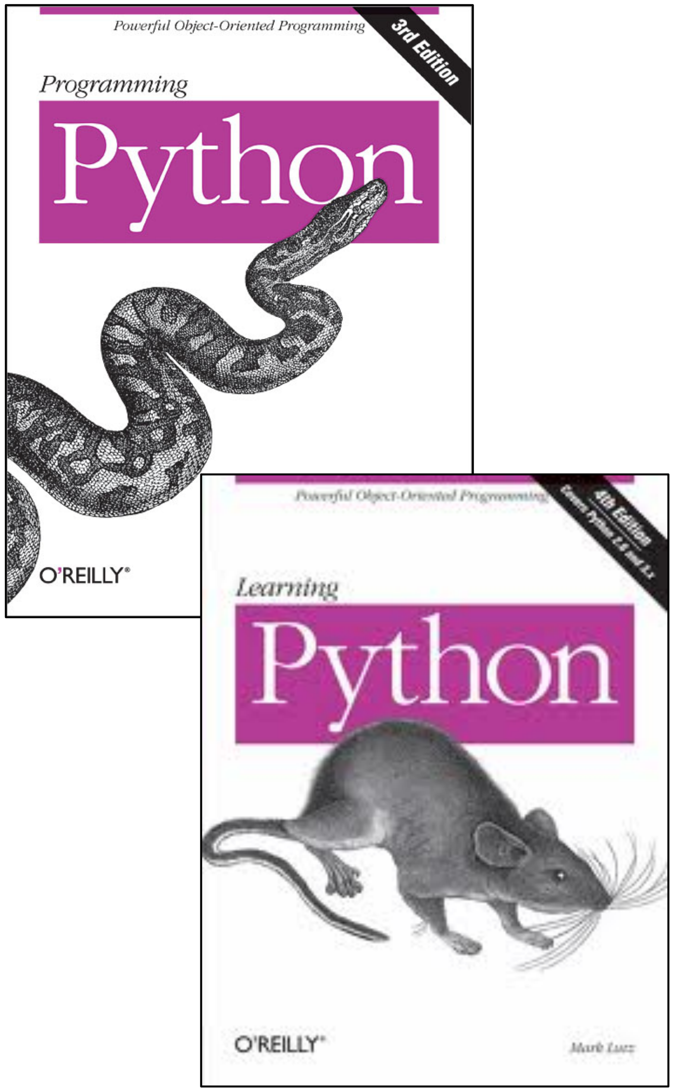
4. Know it when we see it?

Analyzing agent performance

- **Rational agent** is one that does the “right” thing
  - Must define a **performance measure**
    - Costs (penalties) and rewards
  - Chooses an action that maximizes expected score
- Rationality depends on
  1. Success criterion defined by performance measure
  2. “Behavior” of the environment (e.g., can a clean square get dirty again?)
  3. Possible actions
  4. Percept sequence
- Autonomy
  - Rational agents require **learning** to compensate for incorrect or incomplete starting knowledge

Introduction to Python

- Developed by CWI in 1989
- Features
  - Interpreted
  - Dynamic typing
  - Easily readable code blocks
  - Object-oriented—all data is represented by objects or relations between objects
- Good for AI—easy to learn, easy to implement AI concepts
- We will use release 2.x



Writing and running Python

- Interpreter
  - python <file>.py
    - Executes the statements in <file>.py
  - Interactive mode
- Executable scripts
  - #! /usr/bin/env python
- Idle
  - Python IDE packaged with the download
  - Automatic block indention and text highlighting
  - Some EMACS keys work
  - Debugger

Simple Python example

- Let’s start with a (very!) simple example

```
# This is a Python program
x = 37
y = x + 5
print y
```

```
% python simple.py
42
%
```

- Variables need not be declared, but must be assigned
- Line breaks separate statements
- Comments begin with #

Variable assignments, numbers, and strings

- Variable assignment with =

```
>>> x = y = 42
```

- Straightforward math expressions using +, -, *, /, and ( )

```
>>> (50 - 5 * 6) / 4
5
```

- Support for floating point and mixed computation (convert all operands to floating point)

```
>>> 3 * 3.75 / 1.5
7.5
```

- Strings can be enclosed in either single or double quotes
  `“python”` or `‘python’`
- Indexing with []
- Muli-line strings are enclosed in triple quotes or end in \
- Concatenation with + and repetition with *

```
>>> “str”*3
“strstrstr”
```

Data structures and compound data types

- Lists—comma separated values enclosed by [ ]

```
>>> l = ["hello", "world", 42]
["hello", "world", 42]
>>> nl = [[2,3],[4,5]]
```

  - List comprehension—lists resulting from evaluating expressions

  ```
  >>> vec = [2, 3, 4]
  >>> [3*x for x in vec]
  [6, 9, 12]
  ```

- Tuples—group of values separated by commas

```
>>>  t = 1, 5, 9   #tuple packing
(1, 5, 9)
>>> x, y, z = t    #tuple unpacking
>>> t = ()
>>> t = "singleton",
```

- Slices—subset of list (or string)
  - Defined by two indices

```
>>> slicedString = aString(start : end)
>>> slicedList = aList(start : end)
```

Dictionaries

- Unordered sets of *key: value* pairs (Associative arrays)
  - Indexed by unique keys
  - Keys must be immutable types (i.e., strings, numbers)—can use tuples only if they contain immutable elements

```
>>> grades = { }
>>> grades = {“joe”: 93, “sally”: 82}
>>> grades[“bill”] =  87
>>> grades
{“bill”: 87, “sally”: 82, “joe”, 93}
```

- Construct dictionaries from a list of key: value tuples

```
>>> dict( [( “joe”,93), (“sally”, 82), (“bill”, 87)] )
{“bill”: 87, “sally”: 82, “joe”, 93}
```

Control flow and functions

- if statements (conditionals)

```
>>> if x < 0
…             print 'Negative'
…       elif x = 0:
…             print 'Zero'
…       else:
              print 'Positive'
```

- while--continue looping until condition is false

```
>>> a, b = 0, 1
>>> while b < 1000
…             print b
…             a, b = b, a+b
```

- for—iteration over a sequence

```
>>> a = [1, 6, 15]
>>> for x in a:
…             print x
```

  - range() function—creates sequences useful for iteration
- Indentation—required for grouping statements
- Function definitions

```
>>> def fib(n): # Fibonacci series up to n using above while
```

Manipulating directories and files

- Access modules using **`import`** keyword
- OS module—access operating system dependent functionality
  - os.path—module with useful functions for pathnames (e.g., normalize absolute paths, find directories,…)
  - os.getcwd()—returns string which names current directory
  - os.chdir("C:/") or os.chdir("C:\\")—change the current working directory to the specified path
- Reading files
  - open(name, mode)—name is the filename, mode is read (‘r’), write (‘w’), or append (‘a’)
    - Returns File object
    - If no mode is specified defaults to ‘r’

```
>>> myfile = open(“…”)
>>> s = myfile.readline()  # string containing the next line
>>> ss = myfile.readlines() # list containing all lines
```

- Two approaches to reading data structures from files
  - [Yes , Happy] → use `r = pfile.readline()` and then parse it into a list
  - [‘Yes’ , ‘Happy’] → use `p = eval(pfile.readline())`

Classes and object oriented programming

- Class definitions and instantiation

```
>>> class C1: ...
>>> I1 = C1( )
```

- Inheritance

```
>>> class C2(C1): …
```

- Class attributes and methods

```
>>> class C2(C1):
…       data = value
…       def setname(self, who):
…               self.name = who
```

  - Class attributes defined at top level are shared by all instances, but changes to the value only affect the instance
  - self—self reference to current instance
  - Constructor method named __init__
  - Methods can be accessed as unbound (at class level) or bound to an instance

- Overload operators for classes

```
>>> class C1:
…       def __add__(self,other)
…               return C1(self.data + other)
```

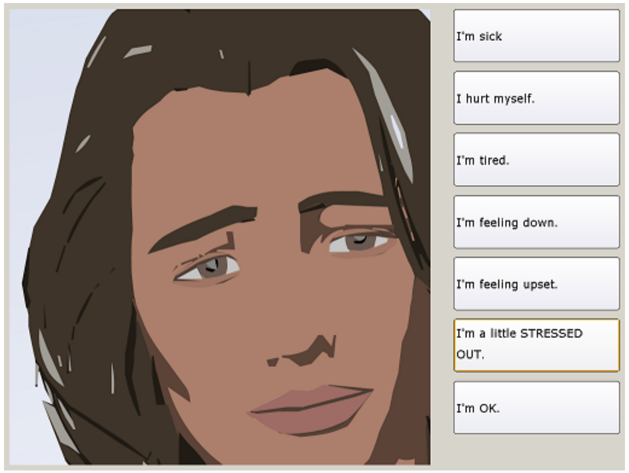
For more help…

- See the **Resources** page on the class website
  - Check out the **example programs** count.py , match.py, oodemo.py and the vacuum agent implemented with Python
- Visit the Python tutorial

Assignment 1
A Relational Agent in Python

A relational agent program in Python

- **Objective:** To get acquainted with Python and the use of production rules.
- **Introduction:** Relational Agents are one kind of AI program, involving animated characters that produce gestures such as smiling, blinking, and nodding, to make them seem more realistic. Prof. Bickmore's research attempts to see if relational agents that show empathy can influence users to improve their health behavior.



- **Project Description:** We will program the gesture selection component of a relational agent. The set of actions will be:
  1. Smile
  2. Frown
  3. Nod
  4. Blink

  Percepts will be a user's communication to the program, consisting of two elements:
  1. content (Positive, Negative, Unsure)
  2. mood (Happy, Sad, Neutral)

  The output is a gesture that the relational agent should perform when responding. The agent's gesture selection strategy is defined as:
    - If happy and either positive or negative, smile
    - If happy and unsure, nod
    - If sad, frown
    - If neutral and positive or negative, nod
    - If neutral and unsure, blink

- **Your Task:** Implement a production-rule driven relational agent with the behavior as above. Input will be a file of 2-element Python tuples, one per line. Your top-level function should be called gestures(path)—the path argument being a path to the input file, suitable as an argument to open.