

6.001: Structure and Interpretation of Computer Programs

- Today
 - The structure of 6.001
 - The content of 6.001
 - Beginning to Scheme

2/5/2007

6.001 SICP

1/56

6.001

- Main sources of information on logistics:
 - General information handout
 - Course web page
 - <http://sicp.csail.mit.edu/>
 - <http://sicp.csail.mit.edu/Spring-2007/>

2/5/2007

6.001 SICP

2/56

Course structure

- Lectures
 - Delivered live here, twice a week (Tuesday and Thursday)
 - Versions of lectures also available on the web site, as audio annotated Power Point. Treat this like a live textbook. Versions are **not identical** to live lecture, but cover roughly same material.
 - Because lecture material is evolving, we **strongly** suggest that you attend live lectures, and use the online lectures as reinforcement.
- Recitations
 - Twice a week (Wednesday and Friday)
 - **For Wednesday, don't go to recitation assigned by registrar: check the web site for your assigned section. If you have conflict, contact course secretary by EMAIL only.**
 - You are **expected** to have attended the lecture (or listened to the online version) before recitation
 - Opportunity to reinforce ideas, learn details, clarify uncertainties, apply techniques to problems
- Tutorials
 - Once a week (typically Monday, some on Tuesday)
 - **You should really be there – we provide a "carrot" to encourage you**
 - Ask questions, participate in active learning setting

2/5/2007

6.001 SICP

3/56

6.001

- Grades
 - 2 mid-term quizzes – 25%
 - Final exam – 25%
 - 1 introductory project and 5 extended programming projects – 40%
 - weekly problem sets – 10 %

BUT YOU MUST ATTEMPT ALL OR COULD RESULT IN FAILING GRADE!!

 - **Participation** in tutorials and recitations – up to 5% bonus points!!

2/5/2007

6.001 SICP

4/56

Contact information

- Web site: <http://sicp.csail.mit.edu/>
- Course secretary
 - Donna Kaufman, dkauf@mit.edu, 38-409a, 3-4624
- Instructor in charge, lecturer
 - Eric Grimson, welg@csail.mit.edu
- Co-lecturer
 - Rob Miller, rcm@csail.mit.edu



2/5/2007

6.001 SICP

5/56

Section Instructors



Prof. Michael Collins



Gerald Dalley



Prof. Berthold Horn



Prof. Peter Szolovits



Dr. Kimberle Koile

2/5/2007

6.001 SICP

6/56

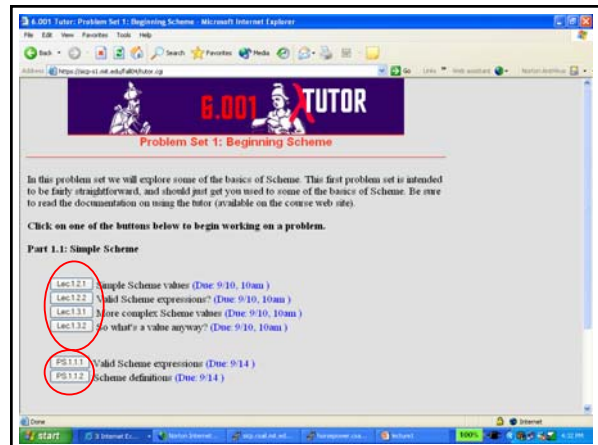
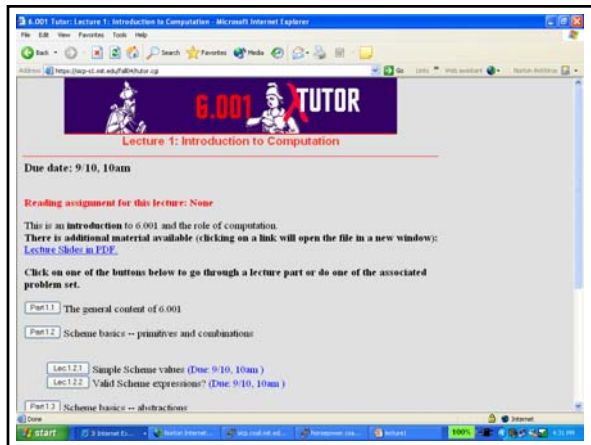
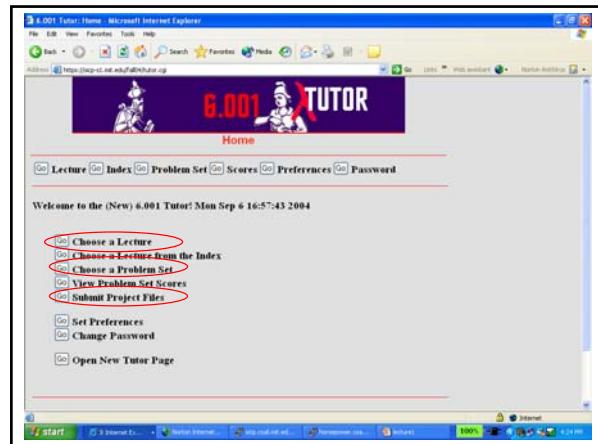
Other logistics

- Problem sets
 - Are released through the online tutor (see website for link – there is a separate link to register for the tutor)
 - Are due electronically on the date posted
 - Includes lecture problems
 - You should **really** try to do these problems before the associated recitation
 - If section instructors find that too many students are coming unprepared, we will change these problems to be due on day of associated recitation!
 - **First one was posted today!**
- Projects
 - First one will be released today
 - Check website for updates

2/5/2007

6.001 SICP

7/56



Other Issues

- Collaboration – Read description on web site
- Use of bibles – See description on web site
- Time spent on course
 - Survey shows 15-18 hours/week
 - Seeking help
 - Lab assistants
 - Other sources – departmental tutoring services, institute tutoring services (ask for help if you think you need it)
- 6.001 Lab – 34-501
- Combination
 - Inner door: **04862***
 - Outer door: **94210** (evenings, weekends)

2/5/2007

6.001 SICP

11/56

Other Issues

- Slides: You have **most** of them.
- Because sometimes...
 - there are answers to problems
 - there are jokes
 - it's good to pay attention

2/5/2007

6.001 SICP

12/56

Getting assigned to a recitation

- We are **NOT** going to use the registrar's recitation assignments
- Please take a few minutes to fill out the sign up sheet
 - Turn in at the end of lecture
- We will post assignments for tomorrow's section later this afternoon on the course web site

2/5/2007

6.001 SICP

13/56

6.001

- Today
 - The structure of 6.001
 - **The content of 6.001**
 - Beginning to Scheme

2/5/2007

6.001 SICP

14/56

What is the main focus of 6.001?

- This course is about **Computer Science**.
- Geometry was once equally misunderstood.
 - Term comes from *ghia & metra* or earth & measure – suggests geometry is about surveying
 - But in fact it's about...
- By analogy, computer science deals with *computation*; knowledge about *how to compute* things
- Imperative knowledge

2/5/2007

6.001 SICP

15/56

Declarative Knowledge

- “What is true” knowledge

\sqrt{x} is the y such that $y^2 = x$ and $y \geq 0$

2/5/2007

6.001 SICP

16/56

Imperative Knowledge

- “How to” knowledge
- To find an approximation of square root of x :
 - Make a guess G
 - Improve the guess by averaging G and x/G
 - Keep improving the guess until it is good enough

Example: \sqrt{x} for $x = 2$.

X = 2	G = 1

2/5/2007

6.001 SICP

17/56

Imperative Knowledge

- “How to” knowledge
- To find an approximation of square root of x :
 - Make a guess G
 - Improve the guess by averaging G and x/G
 - Keep improving the guess until it is good enough

Example: \sqrt{x} for $x = 2$.

X = 2	G = 1
X/G = 2	G = $\frac{1}{2}(1 + 2) = 1.5$

2/5/2007

6.001 SICP

18/56

Imperative Knowledge

- “How to” knowledge
- To find an approximation of square root of x :
 - Make a guess G
 - Improve the guess by averaging G and x/G
 - Keep improving the guess until it is good enough

Example: \sqrt{x} for $x = 2$.

$X = 2$	$G = 1$
$X/G = 2$	$G = \frac{1}{2}(1 + 2) = 1.5$
$X/G = 4/3$	$G = \frac{1}{2}(3/2 + 4/3) = 17/12 = 1.416666$

2/5/2007

6.001 SICP

19/56

Imperative Knowledge

- “How to” knowledge
- To find an approximation of square root of x :
 - Make a guess G
 - Improve the guess by averaging G and x/G
 - Keep improving the guess until it is good enough

Example: \sqrt{x} for $x = 2$.

$X = 2$	$G = 1$
$X/G = 2$	$G = \frac{1}{2}(1 + 2) = 1.5$
$X/G = 4/3$	$G = \frac{1}{2}(3/2 + 4/3) = 17/12 = 1.416666$
$X/G = 24/17$	$G = \frac{1}{2}(17/12 + 24/17) = 1.4142156$

2/5/2007

6.001 SICP

20/56

“How to” knowledge

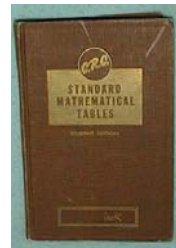
Why “how to” knowledge?

- Could just store tons of “what is” information

2/5/2007

6.001 SICP

21/56



2/5/2007

NUMERICAL TABLES
Table F.2. Five-place Common Logarithms of Numbers*

N	0	1	2	3	4	5	6	7	8	9
1.0	0.0000	0.0004	0.0008	0.0012	0.0016	0.0020	0.0024	0.0028	0.0032	0.0036
1.1	0.0414	0.0427	0.0440	0.0453	0.0466	0.0479	0.0492	0.0505	0.0518	0.0531
1.2	0.0792	0.0804	0.0816	0.0828	0.0840	0.0852	0.0864	0.0876	0.0888	0.0900
1.3	0.1132	0.1143	0.1154	0.1165	0.1176	0.1187	0.1198	0.1209	0.1220	0.1231
1.4	0.1461	0.1471	0.1481	0.1491	0.1501	0.1511	0.1521	0.1531	0.1541	0.1551
1.5	0.1761	0.1771	0.1781	0.1791	0.1801	0.1811	0.1821	0.1831	0.1841	0.1851
1.6	0.2041	0.2051	0.2061	0.2071	0.2081	0.2091	0.2101	0.2111	0.2121	0.2131
1.7	0.2301	0.2311	0.2321	0.2331	0.2341	0.2351	0.2361	0.2371	0.2381	0.2391
1.8	0.2553	0.2563	0.2573	0.2583	0.2593	0.2603	0.2613	0.2623	0.2633	0.2643
1.9	0.2803	0.2813	0.2823	0.2833	0.2843	0.2853	0.2863	0.2873	0.2883	0.2893
2.0	0.3010	0.3020	0.3030	0.3040	0.3050	0.3060	0.3070	0.3080	0.3090	0.3100

22/56

“How to” knowledge

Why “how to” knowledge?

- Could just store tons of “what is” information
- Much more useful to capture “how to” knowledge – a series of steps to be followed to deduce a particular value
 - a recipe
 - called a **procedure**
- Actual evolution of steps inside machine for a particular version of the problem – called a **process**
- Want to distinguish between procedure (recipe for square root in general) and process (computation of specific result); former is often much more valuable

2/5/2007

6.001 SICP

23/56

Describing “How to” knowledge

If we want to describe processes, we will need a language:

- Vocabulary – **basic primitives**
- Rules for writing compound expressions – **syntax**
- Rules for assigning meaning to constructs – **semantics**
- Rules for capturing process of evaluation – **procedures**

15 minutes

2/5/2007

6.001 SICP

24/56

Using procedures to control complexity

This is what we are actually going to spend the term discussing

Goals: Given a specific problem domain, we need to

- Create a set of primitive elements— simple data and procedures
- Create a set of rules for combining elements of language
- Create a set of rules for abstracting elements – treat complex things as primitives

Why abstraction? -- Can create complex procedures while suppressing details

Target: Create complex systems while maintaining: efficiency, robustness, extensibility and flexibility.

2/5/2007

6.001 SICP

25/56

Key Ideas of 6.001

- **Linguistic perspective on engineering design** But no HASS credit!
 - Primitives
 - Means of combination
 - Means of abstraction
 - Means for capturing common patterns

Look at generic elements, but also at how to design for specific problem domain
- **Controlling complexity**
 - Procedural and data abstractions
 - Recursive programming, higher order procedures
 - Functional programming versus object oriented programming
- **Metalinguistic abstraction**
 - Creating new languages
 - Creating evaluators

2/5/2007

6.001 SICP

26/56

6.001

- Today
 - The structure of 6.001
 - The content of 6.001
 - **Beginning Scheme**

2/5/2007

6.001 SICP

27/56

Computation as a metaphor

- Capture descriptions of computational processes
- Use abstractly to design solutions to complex problems
- Use a language to describe processes

2/5/2007

6.001 SICP

28/56

Describing processes

- **Computational process:**
 - Precise sequence of steps used to infer new information from a set of data
- **Computational procedure:**
 - The “recipe” that describes that sequence of steps in general, independent of specific instance
- **What are basic units on which to describe procedures?**
 - Need to represent information somehow

2/5/2007

6.001 SICP

29/56

Representing basic information

- **Numbers**
 - Primitive element – single binary variable
 - Takes on one of two values (0 or 1)
 - Represents one bit (binary digit) of information
 - Grouping together
 - Sequence of bits
 - Byte – 8 bits
 - Word – 16, 32 or 48 bits
- **Characters**
 - Sequence of bits that encode a character
 - EBCDIC, ASCII, other encodings
- **Words**
 - Collections of characters, separated by spaces, other delimiters

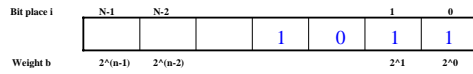
2/5/2007

6.001 SICP

30/56

Binary numbers and operations

- Unsigned integers



$$\sum_{i=0}^{n-1} b_i \cdot 2^i \quad \text{where } b_i \text{ is 0 or 1}$$

$1 + 2 + 8 = 11$

2/5/2007

6.001 SICP

31/56

Binary numbers and operations

- Addition

$$\begin{array}{r} 0 \quad 0 \quad 1 \quad 1 \\ +0 \quad +1 \quad +0 \quad +1 \\ \hline 0 \quad 1 \quad 1 \quad 10 \end{array}$$

$$\begin{array}{r} 10101 \\ + 111 \\ \hline 11100 \end{array}$$

2/5/2007

6.001 SICP

32/56

Binary numbers and operations

- Can extend to signed integers (reserve one bit to denote positive versus negative)
- Can extend to character encodings (use some high order bits to mark characters versus numbers, plus encoding)

2/5/2007

6.001 SICP

33/56

Where *Are* The 0's and 1's?



2/5/2007

6.001 SICP

34/56

Where *Are* The 0's and 1's?

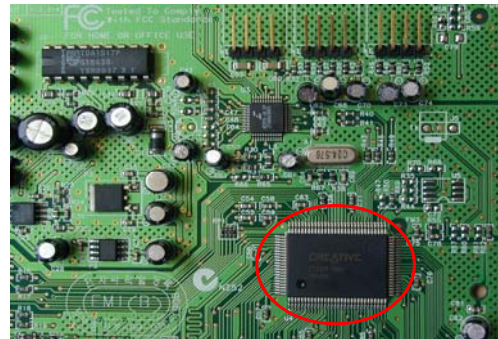


2/5/2007

Replacing a bad tube meant checking among ENIAC's 19,000 possibilities.

35/56

Where *Are* The 0's and 1's?



... we don't care at some level!

- Dealing with procedures at level of bits is **way too low-level!**
- From perspective of language designer, simply need to know the interface between
 - Internal machine representation of bits of information, and
 - Abstractions for representing higher-order pieces of information, plus
 - Primitive, or built-in, procedures for crossing this boundary
 - you give the procedure a higher-order element, it converts to internal representation, runs some machinery, and returns a higher-order element

2/5/2007

6.001 SICP

37/56

Assuming a basic level of abstraction

- We assume that our language provides us with a basic set of data elements ...
 - Numbers
 - Characters
 - Booleans
- ... and with a basic set of operations on these primitive elements, together with a “contract” that assures a particular kind of output, given legal input
- Can then focus on using these basic elements to construct more complex processes

2/5/2007

6.001 SICP

38/56

Our language for 6.001

- Scheme
 - Invented in 1975
- Dialect of Lisp
 - Invented in 1959



Guy Steele Gerry Sussman



John McCarthy

2/5/2007

6.001 SICP

39/56

Rules for describing processes in Scheme

1. Legal expressions have rules for **Syntax** constructing from simpler pieces
2. (Almost) every **expression** has a **value**, which is “returned” when an expression is “**evaluated**”. **Semantics**
3. Every value has a **type**, hence every (almost) expression has a **type**.

2/5/2007

6.001 SICP

40/56

Kinds of Language Constructs

- Primitives
- Means of combination
- Means of abstraction

2/5/2007

6.001 SICP

41/56

Language elements – primitives

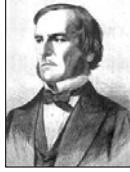
- Self-evaluating primitives – value of expression is just object itself
 - Numbers: 29, -35, 1.34, 1.2e5
 - Strings: “this is a string” “ this is another string with %&^ and 34”
 - Booleans: #t, #f

2/5/2007

6.001 SICP

42/56

George Boole



A Founder

An Investigation of the Laws of Thought, 1854
-- "a calculus of symbolic reasoning"

2/5/2007

6.001 SICP

43/56

Language elements – primitives

- Built-in procedures to manipulate primitive objects
 - Numbers: +, -, *, /, >, <, >=, <=, =
 - Strings: string-length, string=?
 - Booleans: boolean/and, boolean/or, not

2/5/2007

6.001 SICP

44/56

Language elements – primitives

- Names for built-in procedures
 - +, *, -, /, =, ...
 - What is the value of such an expression?
 - + → [#procedure ...]
 - Evaluate by looking up value associated with name in a special table

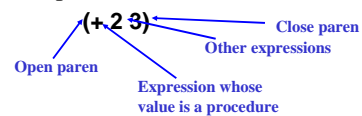
2/5/2007

6.001 SICP

45/56

Language elements – combinations

- How do we create expressions using these procedures?



- Evaluate by getting values of sub-expressions, then applying operator to values of arguments

2/5/2007

6.001 SICP

46/56

Language elements - combinations

- Can use nested combinations – just apply rules recursively

(+ (* 2 3) 4)
→10

(* (+ 3 4)
(- 8 2))
→42

2/5/2007

6.001 SICP

47/56

Language elements -- abstractions

- In order to abstract an expression, need way to give it a name
(define score 23)
- This is a special form
 - Does not evaluate second expression
 - Rather, it pairs name with value of the third expression
- Return value is unspecified

2/5/2007

6.001 SICP

48/56

Language elements -- abstractions

- To get the value of a name, just look up pairing in environment
 - score** → 23
 - Note that we already did this for +, *, ...
(define total (+ 12 13))
(* 100 (/ score total)) → 92
- This creates a loop in our system, can create a complex thing, name it, treat it as primitive

2/5/2007

6.001 SICP

49/56

Scheme Basics

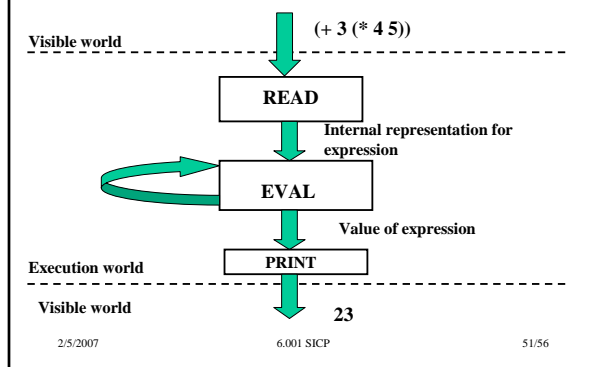
- Rules for evaluation
 - If **self-evaluating**, return value.
 - If a **name**, return value associated with name in environment.
 - If a **special form**, do something special.
 - If a **combination**, then
 - Evaluate all of the subexpressions of combination (in any order)
 - apply the operator to the values of the operands (arguments) and return result

2/5/2007

6.001 SICP

50/56

Read-Eval-Print Loop

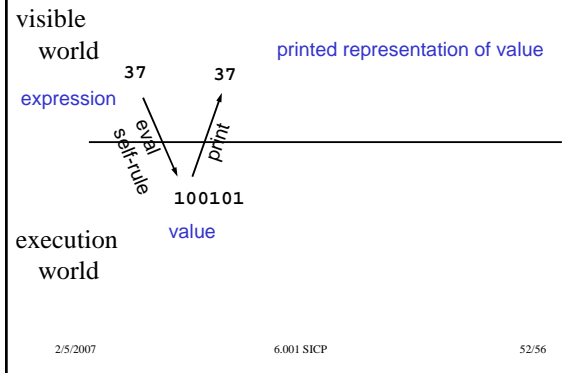


2/5/2007

6.001 SICP

51/56

A new idea: two worlds

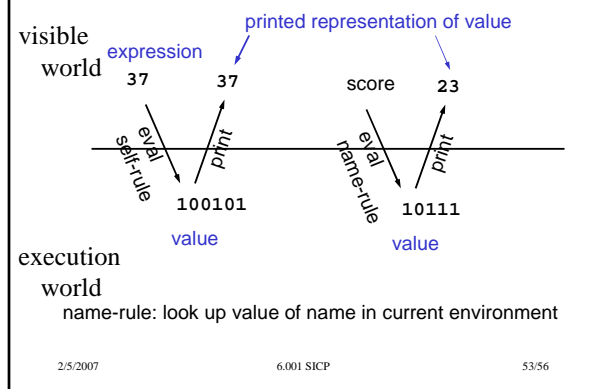


2/5/2007

6.001 SICP

52/56

A new idea: two worlds



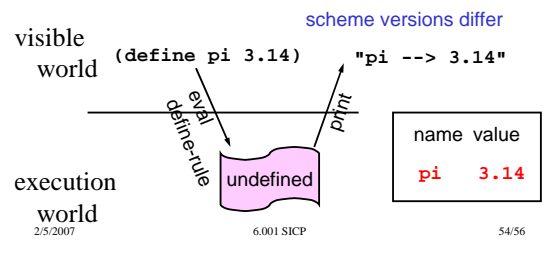
2/5/2007

6.001 SICP

53/56

Define special form

- define-rule:
 - evaluate 2nd operand only
 - name in 1st operand position is bound to that value
 - overall value of the define expression is undefined



2/5/2007

6.001 SICP

54/56

Mathematical operators are just names

```
(+ 3 5)      → 8  
(define fred +) → undef  
(fred 4 6)   → 10
```

• How to explain this?

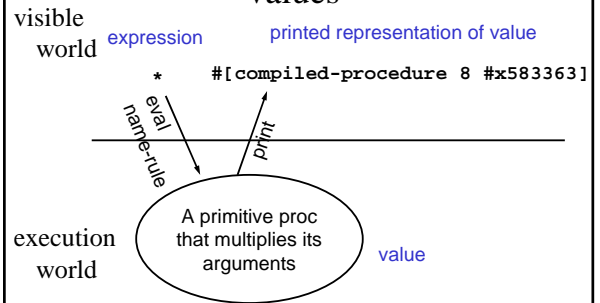
- Explanation
 - + is just a name
 - + is bound to a value which is a procedure
 - line 2 binds the name **fred** to that same value

2/5/2007

6.001 SICP

55/56

Primitive procedures are just values



2/5/2007

6.001 SICP

56/56

Summary

- Primitive data types
- Primitive procedures
- Means of combination
- Means of abstraction – names

2/5/2007

6.001 SICP

57/56