Programming languages

The programming language is the notation that defines the syntax and semantics of programs.

Our programming language is Eiffel.

There are many programming languages, some "general", some "specialized".

Programming languages are artificial notations, designed for a specific purpose (programming).

Object technology

Source: Simula 67 language, Oslo, mid-sixties
Spread very slowly in the seventies
Smalltalk, developed at Xerox PARC in late seventies, made O-O hip by combining it with visual technologies
First OOPSLA conference in 1986 revealed O-O to the masses
Spread quickly in 1990s through O-O languages like Objective C, C++, Eiffel, Java, C#, as well as O-O tools, O-O databases, O-O analysis...
Largely accepted today
Non O-O approaches are also called "procedural".
About Eiffel

First version 1985, constantly refined and improved since
Fully O-O, not a hybrid with other approaches
Focus: software quality, especially reliability, extendibility, reusability
Emphasizes simplicity
Used for mission-critical projects in industry
Based on concepts of “Design by Contract”.
Implementations: from Eiffel Software, Object Tools,
University of Nancy (“SmartEiffel”) 
International standard: ECMA

Some Eiffel-based projects

Axa Rosenberg
Investment management: from $2 billion to >$40 billion
2 million lines of Eiffel

Chicago Board of Trade
Price reporting system
Eiffel + CORBA +
Solaris + Windows + …

Xontech (for Boeing)
Large-scale Eiffel
simulations of missile defense system

Swedish social security: accident reporting & management

So, why use Eiffel?

- Simple, clean O-O model
- Enables you to focus on concepts, not language
- Little language “baggage”
- Development environment (EiffelStudio)
- Portability: Windows / Linux / Mac / VMS & others
- Realism: not an “academic” language

Prepares you to learn other O-O languages, e.g. C++, Java, C# if you need to.
Instructions

Basic operations are called instructions.

Our first example had six instructions:

Paris, display
Louvre, lighten
Line8, spotlight
Paris, build
Paris, equip
Console, show (Line8, origin)

Successive instructions

You may write them one after the other without semicolons:

Paris, display
Louvre, lighten
Line8, spotlight
Paris, build
Console, show (Line8, origin)

You may use semicolons to separate them:

Paris, display; Louvre, lighten
Line8, spotlight; Paris, build
Console, show (Line8, origin)

Style rule

Write one instruction per line
Omit semicolons

If you ever feel it's clearer to have more than one instruction on a line (e.g. in a paper report) use semicolons:

\[ f(x) : g(y) \]
Expressions

An expression is a program element denoting possible run-time values.

Examples:

*Console.show(Line8, origin)*

Also, standard mathematical expressions: \( a + b \).

Definitions

In program texts:

- An instruction denotes a basic operation to be performed during the program’s execution.
- An expression denotes a value used by an instruction for its execution.

Syntax and semantics

An expression, e.g. *Line8, origin* is not a value but denotes future run-time values.

An instruction, e.g. *Paris, display* denotes an operation to be executed at run time.
Syntax and semantics

The syntax of a program is the structure and form of its text.
The semantics of a program is the set of properties of its potential executions.

Syntax is the way you write a program: characters grouped into words grouped into bigger structures.
Semantics is the effect you expect from this program.

Syntax and semantics

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction</td>
<td>Command</td>
</tr>
<tr>
<td>Expression</td>
<td>Query</td>
</tr>
</tbody>
</table>

Syntax structure of a class

class PREVIEW inherit TOURISM

-- Show city info and route:
do
  Paris • display
  Louvre • lighten
end
end
Programming vs natural languages: similarities

- Overall form of texts: succession of words
- Distinction between syntax and semantics
- Some words predefined, others user-defined

Programming vs natural languages: differences

- Power of expression much higher with natural languages
- Precision much higher in programming languages

Programming languages are extensions of mathematical notation

Comments are bits of natural language appearing in programs

Style rule

Use words from natural language (e.g. English, German) for the names you define.

Examples:
- Paris, Line8
- Feature names: show, origin

Eiffel keywords are English words: inherit, do, end...

All single words except elseif
Syntax structure of a class

Specimens

Specimens: a syntactic element; for example:
- A class name, e.g. PREVIEW
- An instruction, e.g. Paris:display
- Any of the boxes on the previous page
- The whole class text!

Specimens may be nested (or embedded)

Delimiters, such as keywords (do, end, ...), semicolons, periods • etc. are not specimens.

Specimens and constructs

A construct is a certain type of syntactic element

Every syntactic element is a specimen of a certain construct

For example:
- display is a specimen of the construct Feature_name
- The class text as a whole is a specimen of the construct Class
Feature declaration
Class names
Comment
Feature body
Instructions
Feature names
Syntax structure

Other representation: abstract syntax tree

Abstract syntax tree
Shows the syntax structure
Specimens only: no keywords or other delimiters
(that's why it's abstract)
Uses the notion of tree as in organizational charts of companies.
Trees that grow down...

Intro. to Programming, lecture 3: Dealing with objects II

Trees in computer science

- Represent hierarchical or nested structures
- Similar to e.g. organizational charts (previous page)
- Pictured top-down or left-to-right

Tree properties

Tree rules:
- Every branch connects two nodes
- Every node can have any number (including none) of outgoing branches
- Every node has at most one incoming branch

Types of node:
- Root: node with no incoming branch
- Leaf: node with no outgoing branches
- Internal node: neither root nor leaf

A tree has exactly one root
(Otherwise it would be a forest)
### Intro. to Programming, lecture 3: Dealing with objects II

**Abstract syntax tree**

- **Root** represents overall specimen (outermost rectangle)
- **Internal nodes** (nonterminals) represent substructures containing specimens themselves
- **Leaves** (terminals) represent specimens with no more nesting

The syntax of a programming language is defined by a set of constructs and the structure of these constructs.

---

**Syntax structure**

```plaintext
class PREVIEW inherit TOURISM

feature explore is
  ~ Show city info and route.
  do
    Paris • display
    Louvre • lighten
  end

end
```
The lower level: lexical structure

The basic elements of a program text are tokens:
- Terminals
  - Identifiers: names chosen by the programmer, e.g. Paris or display
  - Constants: self-explanatory values, e.g. 34
- Keywords, e.g. class
- Special symbols: colon, "\" of feature calls).

Tokens define the language's lexical structure.

Three levels of description

Semantic rules define the effect of programming satisfying the syntax rules
Syntax rules define how to make up specimens out of tokens satisfying the lexical rules
Lexical rules define how to make up tokens out of characters

Lexical rule for identifiers

Identifiers
An identifier starts with a letter, followed by zero or more characters, each of which may be:
- A letter.
- A digit (0 to 9).
- An underscore character "_".

You may choose your own identifiers as you please, excluding keywords.
Style rules

- Always choose identifiers that clearly identify the intended role.
- For features, use full names, not abbreviations.
- For multi-word identifiers, use underscores: `bus_station`.
- Use all upper case for classes: `PREVIEW`.

What we have seen

- Programming language concepts
- Eiffel basics
- Syntax (including lexical level) vs semantics
- Trees
- Tree terminology: root, leaf, node...
- Abstract Syntax Trees (AST)
- Basic lexical elements
- Basic style rules

What to do next

Read chapters 1 to 5

Make sure you know all the terminology introduced so far
End of lecture 3