Course Introduction

Karolinska Institutet
Systems Biology and the Omics Cascade (Course 2143)
High-level Programming Concepts

bioScene

Overview

1. Programming Languages
   - High-level overview of PLs in general
2. Computer Programming
   - Using computers and PLs to process data
3. Paradigms
   - The purpose of the various PLs
4. Example Languages
   - A brief look at Perl, Java, R, SQL

bioScene

Who Am I?

- David Leangen
  - david@leangen.net
  - david.leangen@bioscene.co.jp
- Degree in Electrical Engineering
  - Strong bias towards systems and engineering
  - Disgust for inefficiency
  - Like to understand things from a high-level

bioScene

Goals

1. Understand what a programming language is

2. Get a high-level overview of computer programming

3. Understand how abstractions relate to productivity

4. Understand some of the paradigms behind PLs

5. Get the gist of 4 example languages:

    - Perl, Java, R, SQL

bioScene

Programming Languages

Karolinska Institutet
Systems Biology and the Omics Cascade (Course 2143)
High-level Programming Concepts - Module I

bioScene

First Definition

Programming Language:

A collection of words we use to determine in advance something that should be executed at a later date.

bioScene

Basic Examples

- Programming a VCR to record a hockey game
- Creating a timetable for a system of trains

bioScene

Three Components

1. Programmer
   - Knows what the objective is
   - Writes instructions to achieve the objective
2. System
   - Executes instructions of the program
3. Instructions
   - A common language
     - Understood by both programmer and system

bioScene

Back to our Examples

- VCR Example:
  - Programmer
    1. Person who knows which game to record, when, and which channel
  - System
    1. The VCR Itself
  - Language
    1. Simple system consisting of (start time, end time, channel)

Back to our Examples

- Trains Example:
  - Programmer
    1. Person who knows destinations, peak hours, popular stops...
  - System
    1. Consists of trains, tracks, stations...
  - Language
    1. More complex system consisting of times, stations...

bioScene

Back to our Examples

- Both examples
  - Complex systems
  - Different range of constraints
    1. VCR: end time > start time, channel exists...
    2. Trains: two trains cannot be at same platform simultaneously....
  - Different range of states
    1. VCR: limited (recording, not recording)
    2. Trains: very large number of states and transitions

bioScene

Examples

- Can you think of any more examples?

bioScene

Constraints

- Boundaries for a valid conditions
- May change depending on policies
- Determines which values are “legal”
- Example:
  - A “normal” bank account balance must be > 0
    1. If we have ￥10,000 and try to withdraw ￥20,000 → ERROR
  - If we change the policy to allow for overdrafts
    1. If we have ￥10,000 and try to withdraw ￥20,000 → OK

bioScene

States

- Complete set of properties of the system
- In simple systems or state machines
  - Are well-known
  - Can be determined in advance
- In complex systems
  - Can (for practical purposes) approach infinity
  - Rather than reasoning, apply various techniques
- State validity determined by constraints

bioScene

States

- Example: molecule object

```
Molecule

id:       1
name:     hydrogen
mass:     1.00794
...
```

Second Definition

Programming Language:

> A set of instructions that, given a system with a set of constraints and range of valid states, allows us to determine in advance something that should be executed at a later date.

bioScene

Non-Ambiguity

- PL must be non-ambiguous
- System response to PL must be determined
  - Otherwise programming would be impossible
- For a given state with identical constraints
  - System must respond to instruction same way
    1. If it doesn't respond the same way each time → broken system
    2. When it doesn't respond as intended → "bug"

bioScene

Third Definition

Programming Language:

A set of instructions with a well-defined, non-ambiguous grammar and syntax that, given a system with a set of constraints and range of valid states with predetermined state transitions, allows us to determine in advance something that should be executed at a later date.

bioScene

Computer Programming

Karolinska Institutet

Systems Biology and the Omics Cascade (Course 2143)

High-level Programming Concepts - Module II

bioScene

Computer Programming Language

- Our definition of PL still applies
- System is “something” related to computers

Computer Basics

- CPU: Central Processing Unit
  - "Heart" ("brain") of the system
  - Where "all" instructions are processed
- Each processor has its own language
  - "Machine language" or "Assembly"
  - Moving around bits between registers
  - Very low level
  - Very tedious to work with!

bioScene

Assembly Language

- Example: swap values between registers:

```
les di,[a]
mov ax,[es:di]
les di,[b]
mov bx,[es:di]
mov [es:di],ax
les di,[a]
mov [es:di],bx
```

bioScene

Stop!

- My head hurts: there must be a better way!
- Use a higher-level language
  - Can think with higher-level constructs
  - Don't need to remember each processor
  - The "system" will do the work for me
    1. In the case, the "work" is translating into machine code
    2. The "system" is called a **compiler**

bioScene

Definition: Abstraction

- Abstraction

  Abstraction is the process or result of generalization by reducing the information content of a concept or an observable phenomenon, typically in order to retain only information which is relevant for a particular purpose.

  Ref: http://en.wikipedia.org/wiki/Abstraction

bioScene

No More Assembly!

```
# This Assembly code...
    pushl %ebp
    movl %esp, %ebp
    subl $4, %esp
    movl 8(%ebp),%edx
    addl 12(%ebp), %edx
    movl %edx, -4(%ebp)
    movl -4(%ebp), %eax
    leave

# ... can be written much more easily in C code
int add( int i, int j )
{
    int p = i + j;
    return p;
}
```

bioScene

Typical Abstractions

- Data types
  - Date `Date nextXmas = “Dec 25, 2008”`
  - String `String myName = “David”`
  - Integer `Integer myAge = 36`
  - Boolean `Boolean amYoung = false`
- We’ll discuss higher-level abstractions later

Typical Abstractions

- Each of these abstractions (data types)
  - Consists of several lines of machine code
  - Allows us to think in more "natural" terms
  - Can be expressed in more than one ML
    1. So can be ported to any processor that has a compiler
- By shifting up our level of abstraction
  - Huge productivity gains!

bioScene

Key Point

By using higher-level abstractions, we can make huge productivity gains

bioScene

"High-Level" Languages

- We don't want to program in Assembly
- We will use so-called "high-level" languages
- Examples we will briefly look at:
  - Perl
  - Java
  - R
  - SQL

bioScene

Programming Paradigms

Karolinska Institutet
Systems Biology and the Omics Cascade (Course 2143)
High-level Programming Concepts - Module III

bioScene

Outline

- Overview
- Procedural Programming
- Object Orientation

bioScene

Overview of Paradigms

- Programming Paradigms are:
  - Basic way we think about organising programs
  - Way to organise and interpret information flow
  - Interface between human conceptualisation and computer machine languages

bioScene

Overview of Paradigms

- Language determines the paradigm
  - Examples:
    1. Perl: mostly procedural
    2. R: procedural
    3. Java: mostly OO
    4. Prolog: logic
    5. SQL: for conducting queries
  - A language can do both
    1. Perl can be used for procedural or OO-style programming
    2. Java is fundamentally OO, but can be used procedural-style

bioScene

Procedural Programming

- Much like a recipe or list of instructions

```
If it's a Tuesday
  Get up at 6:30
  Have a shower
  Eat breakfast
  Go to the train station
  Wait for the morning express
  ...
```

- Instructions are processed in sequence

Procedural Programming

- Instructions can be grouped together

```
function prepareForWorkOnTuesday
  Get up at 6:30
  Have a shower
  Eat breakfast
  Go to the train station
  Wait for the morning express
```

- So we can recall them later as a single unit

```
if it's a Tuesday, prepareForWorkOnTuesday
```

bioScene

Procedural Programming

- Each language has its own syntax
  - Usually rules are very strict
  - Must be non-ambiguous
- Usually, programs are written to files

bioScene

Procedural Programming

- Files are compiled or interpreted
  - High-level language turned into machine code
  - Directly understood by the processor
    1. Each processor has a different machine language
    2. Very difficult for humans to read and use directly
  - Programs are invoked by a "user"
    1. "User" is a person or another system

Procedural Programming

- Main Benefits
  - Easier to understand
    1. People think this way more naturally when analysing a problem
    2. Don't necessarily require any special training
  - Less concern for good style
    1. Don't have to worry as much about how parts interact
  - Get small jobs done quicker

bioScene

Procedural Programming

- Main Drawbacks
  - Creating large systems can be more difficult
  - Maintaining large systems can be a nightmare

bioScene

Object Orientation

- "Things" grouped together into **objects**
- Objects can be
  - Nouns: people, cars, links...
  - Nominalized actions: manager, runner, connector
  - More abstract: state, workflow, any concept

bioScene

Object Orientation

- Objects are responsible for themselves
- There is no pre-defined order of execution
- Files "compiled" or "interpreted"
  - Same concept as with procedural languages

bioScene

Object Orientation

- Main Benefits
  - Generally easier to write re-usable code
    1. Responsibilities can be grouped togsether more easily
    2. Recalled later in different contexts
  - Re-usable code means
    1. Larger systems easier to develop
    2. Easier maintenance

Object Orientation

- Main Drawbacks
  - Harder to understand than procedural
  - Difficult to code well
  - Often requires more overhead
  - Not so good for simple programs

bioScene

Example

- Example: students at a conference
  - Design Patterns Explained, A New Perspective on Object-Oriented Design Shalloway, Alan et al.
- Description of problem

Let's say that you were an instructor at a conference. People in your class had another class to attend following yours, but didn't know where it was located. One of your responsibilities is to make sure everyone knows how to get to their next class.

bioScene

Example: Procedural Programming

- We will use "functional decomposition"
  - Divide and conquer
    1. Divide the problem into smaller and smaller parts to a point such that each part becomes manageable
  - We may end up with this algorithm:

```
Get a list of the people in the class
For each person in the list
   Find the next class they are taking
   Find the location of the class
   Find the way to get from your classroom
       to the person's next class
   Tell the person how to get to their class
```

Example: Procedural Programming

- Advantages
  - really easy to discover an algorithm
  - really easy to program
- Disadvantages
  - does not express the real world well
  - not adaptable for change
- Observations
  - the "instructor" is responsible for everything
  - so must know all the details

bioScene

Example: Object Orientation

- We will use "conceptual design"
  - An over-simplified definition of objects:
    1. objects represent the "nouns" in the system
  - Better definition:
    1. an object encapsulates a set of responsibilities
  - Each object is responsible for itself
  - We need to understand the concepts to design

bioScene

Example: Object Orientation

- We will likely end up with something like:

Object	Is responsible for...
Student	Knowing which classroom they are in Knowing which classroom they are to go to next Going from one classroom to the next
Instructor	Telling people to go to the next classroom
Classroom	Having a location
Direction Giver	Given two classrooms, giving directions from one classroom to the next

bioScene

Example: Object Orientation

- Advantages:
  - corresponds better to reality
  - more adaptable to change
- Disadvantages:
  - requires more thought and experience to design
  - probably overkill for such a simple problem

bioScene

Example

- Now imagine our requirements change
  - Suppose now that we need to give special instructions to graduate students who are assisting at the conference. Perhaps they need to collect course evaluations and take them to the conference office before they can go to the next class.

Example

- Procedural Programming
  - This simple change of requirements will cause us to need to alter the first program considerably.
- Object Oriented Programming
  - In the second case, each Student knows what it needs to do, so changes to the program are localised and minimal.

bioScene

Example Languages: Perl, Java, R, SQL

Karolinska Institutet

Systems Biology and the Omics Cascade (Course 2143)

High-level Programming Concepts - Module IV

bioScene

Perl: Overview

- Mostly procedural Language
  - Can be used objectively if you really want to
- Touted as good for "getting the job done"
  - TMTOWTDI
  - Much less overhead than with other languages
  - Many various data types
  - Uses interpreter rather than compiler
  - Not type-safe (i.e. easy to get "working")

bioScene

Perl: Overview

- Particularly good for text processing
- Many various modules available
- Centralised repository: CPAN
- Large, active, and mature community
- Many books, tutorials, and documents

bioScene

Perl + Bioinformatics

- BioPerl
  - http://www.bioperl.org
  - Manipulating sequences
  - BLAST
  - FASTA
  - etc.

bioScene

Java: Overview

- Essentially an OO language
  - Though not 100% "pure" OO like Smalltalk
- Touted as good for "enterprise" systems
  - Type-safe
    1. Errors are caught at compile time
    2. A change to an API is "noticed" by all its users
    3. Can "refactor" easily with the help of an IDE
  - Integrated security features

bioScene

Java: Overview

- Virtual machine (write once, run anywhere)
- Many modules available
- Various OSS projects (Apache et al.)
- Large, active, and mature community
- Plenty of commercial support

bioScene

Java + Bioinformatics

- BioJava
  - http://biojava.org
- Bioclipse
  - http://bioclipse.net/
- Etc.

bioScene

R: Overview

- Procedural language
  - Based on S
- Intended for statistical computing
- Includes graphical libraries for plotting
- Extensibility by using libraries
- Mature language, active community

bioScene

R + Bioinformatics

- BioConductor
  - http://www.bioconductor.org
  - R for bioinformatics
    1. Statistical programming
    2. Annotations
    3. Citations

bioScene

SQL: Overview

- A "query language"
  - No good for general programming
  - Only useful for performing queries on data
  - Based on the relational model
- All relational databases support SQL
- Useful for database programmers
- Probably not so useful for users

bioScene

SQL + Bioinformatics

- BioSQL
  - http://www.biosql.org
  - Collaboration between BioPerl, BioJava...
  - O2RM

bioScene

Gratuitous advice: How to Choose?

- Use the right tool for the right job!
- Generally:
  - For very small tasks: use Perl or R
  - For building systems: use Java
- Ideally:
  - You should not try to build large systems
  - You may want to stick with Perl for your tasks
  - Don’t be afraid to dive in!

bioScene

Course Summary

Karolinska Institutet
Systems Biology and the Omics Cascade (Course 2143)
High-level Programming Concepts

bioScene

We Studies 4 Modules

1. Programming Languages
   - High-level overview of PLs in general
2. Computer Programming
   - Using computers and PLs to process data
3. Paradigms
   - The purpose of the various PLs
4. Example Languages
   - A brief look at Perl, Java, R, SQL

bioScene

Our Goals Were

1. Understand what a programming language is

2. Get a high-level overview of computer programming

3. Understand how abstractions relate to productivity

4. Understand some of the paradigms behind PLs

5. Get the gist of 4 example languages:
    - Perl, Java, R, SQL

bioScene

Programming Language

Our definition:

> A set of instructions with a well-defined, non-ambiguous grammar and syntax that, given a system with a set of constraints and range of valid states with predetermined state transitions, allows us to determine in advance something that should be executed at a later date.

bioScene

Computer Programming

- Instructions processed in CPU
  - Processor uses "machine language" (assembly)
  - Each processor has its own assembly language
  - Very tedious to learn
- We use "high-level" languages
  - Much, much more efficient than assembly
  - Allows to think at a higher level of abstraction
  - Do not have to rewrite for each processor

bioScene

Abstractions

- Using higher-level abstract concepts
  - Can provide a huge increase in productivity
- Use right level of abstraction for the job!

bioScene

Paradigms

- Way of thinking behind a language
- We looked at two
  - Procedural
    1. Mostly sequential processing
    2. Much like a recipe of instructions
  - Object Oriented
    1. System comprised of objects
    2. Objects encapsulate responsibilities

bioScene

Example Languages

- Perl
  - Procedural
  - Large, active and mature community
  - Many modules available
  - Good for “getting the job done”
  - BioPerl for bioinformatics (www.bioperl.org)

bioScene

Example Languages

- Java
  - OO
  - Large, active and mature community
  - Many third-party projects / frameworks available
  - Good for large/entreprise systems
  - BioJava for bioinformatics (www.biojava.org)

bioScene

Example Languages

- R
  - Procedural
  - Active and mature community
  - Excellent for statistical programming
  - BioConductor for bioinformatics (www.bioconductor.org)

bioScene

Example Languages

- SQL
  - Useless for general programming
  - Used for conducting queries
  - For relational databases only
  - Try to avoid using SQL if possible
  - BioSQL for bioinformatics (www.biosql.org)

bioScene

終了

Tack så mycket!

bioScene

Modelling

Karolinska Institutet
Systems Biology and the Omics Cascade (Course 2143)
High-level Programming Concepts - Module V

bioScene

Idea to Ponder

- Which seems simpler?

$$(x-x_0)^2+(y-y_0)^2+(z-z_0)^2=r^2$$

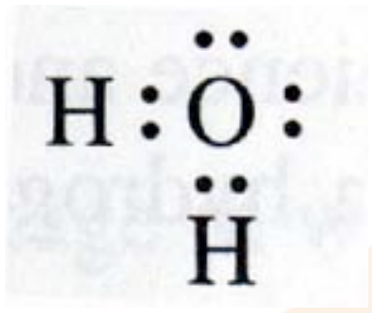
$$(r,\ \theta,\ \varphi)$$

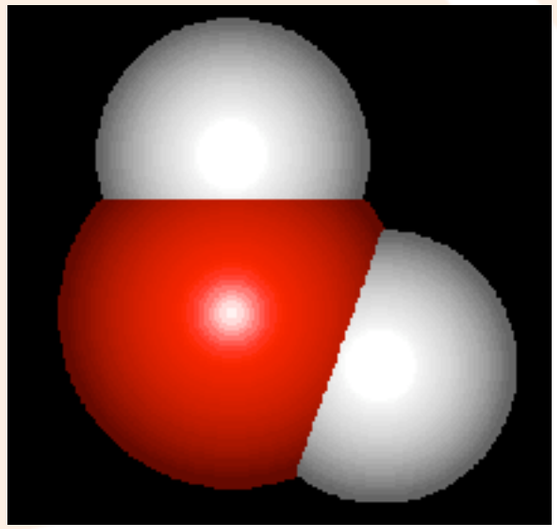
Overview

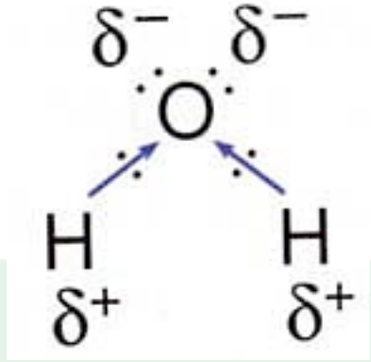
- Models are conceptualisations of reality
- Make concepts manageable in our minds
- Model can vary depending on objective

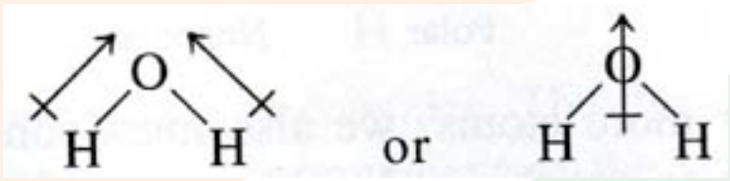
bioScene

Example: Water









Source: http://witcombe.sbc.edu/water/chemistrystructure.html

bioScene

Models and Programming

- Independent of programming language
- Exists no matter what we do
  - People think in models
  - Not just computer science / software devel
- Simplicity depends on the model
  - Some models allow “easy” development
  - Some models work against development

bioScene

Why Models?

- Writing code not difficult
- With some effort, bug-free code possible
- Difficulty is developing a complex system:
  - That has many collaborators
  - That is in a specialized domain
  - For which time and budget are not unlimited
  - That competes for customers/grants
- In other words, any real-life project!

bioScene

Benefits of Models

- Having a clean model helps to:
  - improve communications between collaborators
  - keep project focus
  - bridge domain experts and system developers
  - allow everybody to speak the same language
  - comprehend a very complex system
  - allow to abstract detailed (low-level) concepts

bioScene

Conceptual Layers

- From bits to DSLs
  - hardware: manipulates electronic signals
  - operating system: manipulates the hardware
  - programming languages, compiles, etc:
    uses the OS/hw to manipulate information
  - libraries: groups of programming functionality
  - systems: groups of libraries
  - applications: use of systems in a specific context
  - “documents”: creations by authors using an app

bioScene

Conceptual Layers

- Lowest level: electronic signals
- Highest level: abstract concepts
  - in a particular domain
  - only truly understood by domain experts
  - ideally no need to worry about lower layers

bioScene

Role of Domain Experts (i.e. you!)

- Manipulate information
  - In a domain of your expertise
  - Using a system that understands the domain
  - Using only concepts from your domain
  - No need to worry about how the system works
- Unfortunately, we're not there yet
  - You need to roll up your sleeves and get dirty

bioScene