G4120: Introduction to Computational Biology

Oliver Jovanovic, Ph.D.
Columbia University
Department of Microbiology

Copyright © 2006 Oliver Jovanovic, All Rights Reserved.
History of Programming Languages

1831  Lady Ada Lovelace writes the first computer program, for Charles Babbage's Analytical Engine.

1936  Alan Turing develops the theoretical concept of the Turing Machine, forming the basis of modern computer programming.

1945  John von Neumann develops the theoretical concepts of shared program technique and conditional control transfer.

1946  Plankalkul, the first formal computer language, is developed by Konrad Zuse, a German engineer, which he later applies to, among other things, chess.

1949  Short Code, the first computer language actually used on an electronic computer, appears.

1951  A-O, the first widely used compiler, is designed by Grace Hopper at Remington Rand.

1954  FORTRAN (FORmula TRANslating system) language is developed by John Backus at IBM for scientific computing.

1958  ALGOL, the first programming language with a formal grammar, is developed by John Backus for scientific applications.

1958  LISP (LISt Processing) language is created by John McCarthy of MIT for Artificial Intelligence (AI) research.

1959  COBOL is created by the Conference on Data Systems and Languages (CODASYL) for business programming, and becomes widely used with the support of Admiral Grace Hopper.

1964  BASIC (BEGINner’s ALL-purpose Symbolic Instruction Code) is created by John Kemeny and Thomas Kurtz as an introductory programming language.

1965  Structured programming is defined by Edsger Dijkstra.

1968  Pascal is created by Niklaus Wirth as a teaching language.

1972  PROLOG (Programming Logic) is developed as result of logic theorem research. It has become the most generally used logic programming language, often used in developing expert systems.

1972  Smalltalk, which becomes the first popular object oriented programming language, is developed at Xerox PARC by Alan Kay.
<table>
<thead>
<tr>
<th>Year</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>1972</td>
<td>C is created by Dennis Ritchie at Bell Labs for programming the Unix operating system. It is fast, widely used, and forms the basis of many other current procedural languages.</td>
</tr>
<tr>
<td>1975</td>
<td>Bill Gates and Paul Allen write the first version of Microsoft BASIC.</td>
</tr>
<tr>
<td>1978</td>
<td>AWK, a text-processing language named after the designers, Aho, Weinberger, and Kernighan, appears. It is often used with the sed stream editor to manipulate text data in Unix.</td>
</tr>
<tr>
<td>1979</td>
<td>SQL (Structured Query Language) is developed at IBM based on work to simplify access to data stored in a relational database. It has become the most widely used database language.</td>
</tr>
<tr>
<td>1982</td>
<td>PostScript, a language for graphics printing and display, appears.</td>
</tr>
<tr>
<td>1983</td>
<td>C++, an object-oriented version of the C programming language, appears, based on earlier work on “C with Classes”. It is often used for large projects that require speed.</td>
</tr>
<tr>
<td>1986</td>
<td>Objective C is a Smalltalk influenced object-oriented version of C. It first became widely used as the development language for NeXTstep, and is currently the principle programming language for Mac OS X.</td>
</tr>
<tr>
<td>1987</td>
<td>Perl (Practical Extraction and Reporting Language) is developed by Larry Wall after he finds Unix text utilities limiting. It has become popular as a jack-of-all trades language, and in computational biology applications.</td>
</tr>
<tr>
<td>1991</td>
<td>Python, a simple functional and object oriented language, is developed by Guido Van Rossum. It is often used for rapid development, and is well suited for computational biology applications.</td>
</tr>
<tr>
<td>1991</td>
<td>Visual Basic is developed by Alan Cooper and Microsoft to allow for easy visual creation of Windows applications.</td>
</tr>
<tr>
<td>1994</td>
<td>PHP (Personal Home Page tools) was a simple web server scripting language developed by Rasmus Lerdorf that has developed into a useful language for rapidly developing web applications.</td>
</tr>
<tr>
<td>1995</td>
<td>Java, a simplified version of C++, originally developed by Sun Microsystems to control consumer appliances, is repurposed for web development. It has become popular for writing cross-platform and web applications.</td>
</tr>
<tr>
<td>1995</td>
<td>Ruby, a simple and elegant object oriented programming language, is developed by Yukihiro Matsumoto.</td>
</tr>
</tbody>
</table>
Markup Languages for Bioinformatics

**HTML**

HyperText Markup Language (HTML) is the markup language used by Web pages, derived from Standard Generalized Markup Language (SGML). It was originally developed in 1990 by Tim Berners-Lee while at CERN, for the purpose of sharing scientific data, but has since become the basis of the World Wide Web. It is not a programming language, rather a markup language, which makes it possible to freely present data on the Web. A Web server, such as Apache, serves up the content of a Web page using HyperText Transfer Protocol (HTTP) to a Web browser such as Safari or Internet Explorer, which presents it to the end user in a readable manner. The actual content of a Web page is written in HTML, and the raw HTML source code can be viewed using a Web browser’s View Source command.

[http://www.w3.org/MarkUp/](http://www.w3.org/MarkUp/)

**XML**

The eXtensible Markup Language (XML), is, like HTML, derived from SGML. It was introduced in 1998 as a simpler alternative to SGML for describing complex data. It is not a programming language, but a markup language, which makes it possible to mark up complex data, such as biological data, in a manner that programming languages can easily interpret. It is royalty free, and has been adopted as the data description language of choice by a number of genomics and bioinformatics projects, including the Berkeley Drosophila Genome Project, the Taxonomic Markup Language, the Distributed Annotation System, and the NCI caCORE and caBIO projects.

[http://www.w3.org/XML/](http://www.w3.org/XML/)
Structured Programming

In *structured programming*, programs are created using combinations of four constructs: (1) instruction sequences, (2) branches, (3) loops and (4) modules. The program uses these constructs to perform certain operations on data, which it can input and output.

**Instruction Sequence**
A sequential series of instructions.

**Branch**
A branch, also known as a conditional construct, occurs whenever a program’s flow can divide into two or more streams, depending on whether a particular condition is true or false, such as whether a stop codon has been reached or not.

**Loop**
A loop repeats an instruction or series of instructions a variable number of times, which can be controlled by a test, such as whether the end of a DNA sequence has been reached.

**Modules**
Modules are a way to combine several operations (consisting of one or more of the other three constructs) into a single, reusable component. That component can then be reused throughout the program, or even used by other programs.
Program Elements
A program consists of two elements: (1) data definitions and (2) statements.

Data Definitions
Defines what type of data is used: integers, characters, dates, etc. Types can be static (a variable is always associated with a fixed type) or dynamic, strong (type is known at all times) or weak.

Statements
A program’s statements are typically broken down by function: initialization (preparing the program for its main function), core functionality (the program's main function) and finalization (cleaning up after the program has run).

Batch Oriented Programs
These are programs that are normally started from a command line (or run automatically by a scheduler such as cron). A batch program can simply consist of a text file with a list of programs it runs, or be more complex. When started, a batch program typically initializes the data inside it, reads in what data is specified as input, processes it, and outputs the result.

Event Driven Programs
There are programs that react to certain events sent to it by the operating system. This is typical of graphical user interfaces (GUIs), where an event might be a MouseUp (user moving the mouse up), or MouseClick (user clicks the mouse), which the program then responds to.
Programming Languages

Database Languages
A language tied closely to a database which allows you to easily make queries from a client machine to a database server on which the data is stored.
Example: SQL

Scripting Languages
A simple language that uses a syntax close to a natural language and sends commands to the operating system or other programs when executed.
Example: AppleScript, PHP, Perl

Procedural Languages
A fully featured programming language in which variables can keep changing as the program runs. Most commonly used programming languages are procedural.
Examples: C, Pascal, Perl

Logical Languages
These languages are collections of logical statements and questions.
Examples: Prolog, custom expert systems

Object Oriented Programming Languages
In an object oriented programming (OOP) language, data and functions are encapsulated in objects. An object is a particular instance of a class. Each object can contain different data, but all objects belonging to a class have the same functions or methods. Objects can restrict or hide access to data within them.
Examples: C++, Objective C, Python, Java, Ruby
Perl
The Practical Extraction and Report Language (PERL) is currently the most heavily used programming language in bioinformatics. It is particularly adept at handling arbitrary strings of text and detecting patterns within data, which makes it particularly well suited to working with protein and DNA sequences. In addition, it features a very flexible grammar which allows one to write in a variety of syntaxes, ranging from simple to complex. Perl has been widely used in genomics, including by the human genome project and TIGR. It is distributed under a free open source Artistic License and has become widely adopted by the open source programming community, resulting in numerous useful add on modules for Perl.
http://www.perl.org/

PHP
PHP originated as a set of simple scripting tools called Personal Home Page (PHP), but has become a powerful general-purpose free open source scripting language that is particularly well suited for Web development. PHP code can be embedded into HyperText Markup Language (HTML) and processed by a hypertext preprocessor module for a web server to create dynamic Web pages with minimal effort. Yahoo runs on PHP.
http://www.php.net/

C
The C programming language is one of the oldest programming languages still in wide use. A compiled C program offers excellent performance, and the syntax used by C has influenced most other programming languages in current use.
http://www.lysator.liu.se/c/
Object Oriented Programming Languages for Bioinformatics

Python
A simple object oriented scripting language that is well suited for developing bioinformatics applications and available under a free open source license. It is particularly easy to read and understand, and has been used in a number of structural biology applications.
http://www.python.org/

Ruby
Ruby is a simple and particularly elegant object oriented programming language developed and made available for free by Yukihiro Matsumoto under an open source license. It is designed to easy to use and understand, and is slowly gaining popularity for use in bioinformatics applications. A framework called Ruby on Rails makes it particularly well suited for web development.
http://www.ruby-lang.org/

Java
Java is a powerful object oriented cross-platform programming language developed and made available for free by Sun. It was originally developed for controlling consumer appliances, but repurposed for web development, then expanded. It is particularly well suited for developing complex projects. Although it is simpler than C++, the object oriented version of C, it still takes significant effort to master, but is very powerful, and has been used in a number of major bioinformatics projects.
http://java.sun.com/
Example Perl Program

1) Open TextWrangler, open a new document and type the following source code:

```perl
#!/usr/bin/perl
print ("Who's there?\n");
```

2) Save the plain text file in your home directory as `hello.pl`.

3) Open Terminal, and type `./hello.pl`. It should not execute. Type `ls -l` to check the permissions.

4) Type `chmod 755 hello.pl`, then type `ls -l` again to see how the permissions changed. The permissions for `hello.pl` should now allow it to be executed.

5) Type `./hello.pl`, `~/hello.pl` or `perl hello.pl` to execute the program.
Example Python Program

1) Open a new document in TextWrangler and type the following source code:

```python
#!/usr/bin/python
print "Who's there?"
```

2) Save the file in your home directory as `hello.py`.

3) In Terminal, type `chmod 755 hello.py` to allow for executable permissions.

4) Type `.\hello.py`, `~\hello.py` or `python hello.py`.

5) Python can be run directly from the command line as well. To do so, type `python` in Terminal.

6) At the Python prompt (``>>>'`), type `print "Who's there?"`

7) Press Control and D to exit (this is the EOF key).
Example C Program

1) Open a new document in TextWrangler and type the following source code:

```c
main ()
{
    printf("Who's there?\n");
}
```

2) Save the file in your home directory as `hello.c`.

3) The source code must then be compiled to run. We can use the gcc C compiler to do this by typing `gcc hello.c` in Terminal.

4) This creates a compiled executable program named `a.out` by default. Execute the newly created compiled program by typing `./a.out` or `~/a.out`.

5) Note that unlike Perl, which interprets the `hello.pl` source code directly, in C we have to first compile it to an executable program (the compiler automatically set the permissions of the `a.out` file to be executable), then run the compiled program.
Programming in Perl

Statement (;)
A statement in Perl is the basic unit of execution, and ends in a semicolon.

Block ({ })
A block is a series of statements enclosed by curly braces.

Comment (#)
A comment is anything which starts with a number sign. It is not executed, but rather used to explain what the code is doing, i.e. # this statement prints from 1 to 100

Conditional Constructs (if, else, elsif)
Conditional constructs use code that only executes under certain conditions. The if, else and elsif statements are the most common form of branch control in Perl, allowing you to execute another statement or block depending on whether certain conditions have or have not been met, i.e. if ($x > 0) { printf ("x is a positive number.\n"); }

Loops (for, foreach, while)
A loop repeatedly executes a specific set of statements until a particular condition is reached, which allows for rapid iteration. The for, foreach and while statements are commonly used in Perl loops, i.e. for ($x = 0; $x <= 99; $x++) { print "$x\n"; } print "$x\n";

Subroutines (sub)
To simplify complex code, it is possible to refer to a set of statements as a subroutine, using the sub declaration. This allows them to be repeatedly referred to in a concise manner, instead of having to write out the complete set of statements each time they need to be used.
Variables in Perl

Variables are one of the most useful features of programming languages, allowing a name to be associated with a stored data value, such as a string of text or a number, which can change as the program executes. Perl does not require variables to be formally declared, as opposed to some other programming languages, and uses three kinds of variables: (1) scalars, (2) arrays and (3) hashes.

Scalar ($)
A scalar variable is indicated by a $, and can store a number or a string, i.e.
$sequence = "GCATTTGTGAGACCCCGTACGTAG";

Array (@)
An array variable is indicated by an @, and can store multiple values in an ordered list of data, i.e. @rna = ("G", "C", "A", "U"); The first element in an array is numbered 0, the second 1, the third 2, etc, and a value can be retrieved by specifying its position, i.e.
$value = $rna[3];

Hash (%)
A hash variable is indicated by a %, and is also known as an associative array because it associates a key with each value stored in it, i.e. %stop = (amber => AUG, ochre => UAA, opal => UGA);. The value can then be retrieved from the hash using the appropriate key, i.e. $value = $stop{"amber"};
It is possible to do sophisticated pattern matching and replacement in Perl using regular expressions and transliteration.

The following program uses a regular expression to search for opal stop codons in a case insensitive manner in DNA:

```perl
#!/usr/bin/perl
$sequence = "GCATTTGTGAGACCCGCGTACGTAG";
if ($sequence =~ m/TGA/i) { printf ("Opal stop codon found.\n")}; }
```

The following program uses transliteration to count the number of Gs and Cs in a DNA sequence:

```perl
#!/usr/bin/perl
$sequence = "GCATTTGTGAGACCCGCGTACGTAG";
$countGC = ( $sequence =~ tr/GC// );
printf ("There are $countGC Gs and Cs in the sequence.\n");
```

The following program uses transliteration to convert a DNA sequence to an RNA sequence:

```perl
#!/usr/bin/perl
$sequence = "GCATTTGTGAGACCCGCGTACGTAG";
$sequence =~ ( $sequence =~ tr/T/U/ );
printf ("The sequence converted to RNA is $sequence\n");
```
By combining conditional execution and loops with regular expressions, it is possible to build sophisticated parsers in Perl which will parse complex data and extract useful information from it. One of the most common uses of such parsing in bioinformatics is to reformat or extract useful information from raw BLAST output.

The Perl program that follows is a BLAST parser that cleans up and reformats raw BLAST output into a tab separated value text file which can then be imported and viewed as rows and columns of data in a spreadsheet or database.
#!/usr/bin/perl -w

#blastconv.pl
#Reads in BLAST align (normal or Microbial) files, returns tab seperated values in new file by deleting unwanted filler.
#Returns TSV for GI, description, length, identities, frame, subjectStart, subjectEnd
#Usage: perl blastconv.pl, then enter filename   (returns filename.converted)
#To prepare, trim top and bottom of BLAST align file, add ">gi|" to end (can add this to program if done often).
#Copyright (c) 2002 Oliver Jovanovic

print "Enter name of BLAST align file to convert: ";
$inFileName = <STDIN>;
chomp $inFileName;
open (INTEXTFILE, $inFileName);
@INTEXT = <INTEXTFILE>;
$textFile = join ( '', @INTEXT);        #catenate all lines
$textFile =~s/
//g;                    #remove all newline characters
$textFile =~s/>gnl\||>gi\|/
/g;                #replace all ">gi|" (GI) or ">gnlI" (Microbial) with newline
$textFile =~s/\|\S+\s/\t/g;                     #replace all "|... " with tab (GI)
$textFile =~s/\|/\t/g;                          #replaces all "|" with tab (for Microbial) (leaves GN)
$textFile =~s/\s+Length\s=\s/\t/g;              #replace all "   Length = " with tab (length)
$textFile =~s/Score.+Identities\s=\s/\t/g;    #replace all " Score...Identities = " with tab (identities)
$textFile =~s/\sPositives.+Frame\s=\s/\t/g;    #replace all ", Postives...Frame = " with tab (frame)
$textFile =~s/Query.+?(?=Sbjct:\s\d+)/\t/g;     #replace all Query..(minimal)..(Sbj) with tab
$textFile =~s/\s(?=\d+\n)/\t/g;                 #replace space before subjectEnd with tab
while ($textFile =~/Sbjct:/g)  {                        #while part of below pattern being removed exists anywhere in string
    $textFile =~s/Sbjct:\s\d+\s.+\($1\d+\n)/$1\2/;     #remove "Sbjct: " from subjectStart and junk between it and subjectEnd
}
$textFile = -s/\+\ /g;                                 #clean up multiple spaces in description (replace with one)

open (OUTTEXTFILE, ">".$inFileName.".converted");
print OUTTEXTFILE $textFile;
close (INTEXTFILE);
close (OUTTEXTFILE);
Perl Resources

Learning Perl
A site for people learning Perl. Features an online library, including a free PDF version of Simon Cozens’ Beginning Perl book (http://www.perl.org/books/beginning-perl/), answers to commonly asked questions, and many useful links.

http://learn.perl.org/

CPAN (Comprehensive Perl Archive Network)
A large searchable collection of Perl scripts, modules and documentation. A Perl module, CPAN.pm, can be used to download and install Perl software from the CPAN archive.

http://www.cpan.org/

BioPerl
A toolkit of Perl modules useful for building bioinformatics solutions in Perl. The toolkit is built in an object-oriented manner so that many modules depend on each other to achieve a task. BioPerl includes a number of Perl routines and modules for performing sequence analysis, parsing sequence files, and parsing the results of bioinformatics applications such as BLAST.

http://www.bioperl.org/
References

Programming Books

*Developing Bioinformatics Computer Skills* by Cynthia Gibas & Per Jambeck
*Beginning Perl for Bioinformatics* by James Tisdall
*Mastering Perl for Bioinformatics* by James Tisdall
*Genomic Perl* by Rex A. Dwyer
*Learning Perl* by Randal L. Schwartz & Tom Christiansen
*Elements of Programming with Perl* by Andrew L. Johnson
*Learn to Program Using Python* by Alan Gauld
*The Quick Python Book* by Daryl Harms & Kenneth McDonald
*Programming Ruby* by Dave Thomas

Course Evaluation

[http://microbiology.columbia.edu/icb/evaluation.html](http://microbiology.columbia.edu/icb/evaluation.html)

Course Links