



# Essential Computing for Bioinformatics

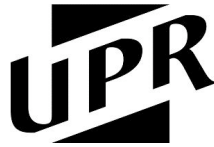
## First Steps in Computing: Course Overview

MARC: Developing Bioinformatics Programs  
July 2009

**Alex Ropelewski**  
PSC-NRBSC

**Bienvenido Vélez**  
UPR Mayaguez

Reference: How to Think Like a Computer Scientist: Learning with Python



- The following material is the result of a curriculum development effort to provide a set of courses to support bioinformatics efforts involving students from the biological sciences, computer science, and mathematics departments. They have been developed as a part of the NIH funded project “Assisting Bioinformatics Efforts at Minority Schools” (2T36 GM008789). The people involved with the curriculum development effort include:
- Dr. Hugh B. Nicholas, Dr. Troy Wymore, Mr. Alexander Ropelewski and Dr. David Deerfield II, National Resource for Biomedical Supercomputing, Pittsburgh Supercomputing Center, Carnegie Mellon University.
- Dr. Ricardo González Méndez, University of Puerto Rico Medical Sciences Campus.
- Dr. Alade Tokuta, North Carolina Central University.
- Dr. Jaime Seguel and Dr. Bienvenido Vélez, University of Puerto Rico at Mayagüez.
- Dr. Satish Bhalla, Johnson C. Smith University.
- Unless otherwise specified, all the information contained within is Copyrighted © by Carnegie Mellon University. Permission is granted for use, modify, and reproduce these materials for teaching purposes.
- Most recent versions of these presentations can be found at <http://marc.psc.edu/>

## Outline

- Course Overview
- Introduction to Programming (Today)
  - Why learn to Program?
  - The Python Interpreter
  - Software Development Process
  - Numbers, Strings, Operators, Expressions
- Control structures, decisions, iteration and recursion

## Course Overview

---

- Essential Computing for Bioinformatics
  - Course Description
  - Educational Objectives
  - Major Course Modules
  - Module Descriptions

This course provides a broad introductory discussion of essential computer science concepts that have wide applicability in the natural sciences. Particular emphasis will be placed on applications to Bioinformatics. The concepts will be motivated by practical problems arising from the use of bioinformatics research tools such as genetic sequence databases. Concepts will be discussed in a weekly lecture and will be practiced via simple programming exercises using Python, an easy to learn and widely available scripting language.

- Awareness of the mathematical models of computation and their fundamental limits
- Basic understanding of the inner workings of a computer system
- Ability to extract useful information from various bioinformatics data sources
- Ability to design computer programs in a modern high level language to analyze bioinformatics data.
- Experience with commonly used software development environments and operating systems
- Experience applying computer programming to solve bioinformatics problems

Module	Lecture	MARC Lecture
First Steps in Computing: Course Overview	1	
Using Bioinformatics Data Sources	2	
Mathematical Computing Models	3	5
High-level Programming (Python): Flow Control	6	2
High-level Programming (Python): Container Objects	7	3
High-level Programming (Python): Files	8	4
High-level Programming (Python): BioPython	9	



## Main Advantages of Python

---

- Familiar to C/C++/C#/Java Programmers
- Very High Level
- Interpreted and Multi-platform
- Dynamic
- Object-Oriented
- Modular
- Strong string manipulation
- Lots of libraries available
- Runs everywhere
- Free and Open Source
- Track record in bioinformatics (BioPython)



## Using Bioinformatics Data Sources

---

### Goal: Basic Experience

- Searching Nucleotide Sequence Databases
- Searching Amino Acid Sequence Database
- Performing BLAST Searches
- Using Specialized Data Sources

Reference: *Bioinformatics for Dummies (Ch 1-4)*

IDEA: How can we expedite data collection and analysis?  
... writing programs to automate parts of the process.



- What is Computing?
- Mathematical Models of Computing
  - Finite Automata
  - Turing Machines
- The Limits of Computation
- Church/Turing Thesis
- What is an Algorithm?
- Big O Notation



## High-Level Programming (Python)

---

### Goal: Knowledge and Experience

- Downloading and Installing the Interpreter
- Values, Expressions and Naming
- Designing your own Functional Building Blocks
- Controlling the Flow of your Program
- String Manipulation (Sequence Processing)
- Container Data Structures
- File Manipulation



## CS Fundamentals will be Interleaved Throughout the Course

---

- Information Representation and Encoding
- Computer Architecture
- Programming Language Translation Methods
- The Software Development Cycle
- Fundamental Principles of Software Engineering
- Basic Data Structures for Bioinformatics
- Design and Analysis of Bioinformatics Algorithms



## Why Learn to Program?

US Department of Labor, Bureau of Labor Statistics  
*Engineers, Life and Physical Scientists and Related Occupations.*  
*Occupational Outlook Handbook, 2008-09 Edition.*

*Biological scientists “...usually study allied disciplines such as mathematics, physics, engineering and computer science. **Computer courses are beneficial for modeling and simulating biological processes, operating some laboratory equipment and performing research in the emerging field of bioinformatics**”*

## Why Learn to Program?

- Need to compare output from a new run with an old run. (new hits in database search)
- Need to compare results of runs using different parameters. (Pam120 vs Blosum62)
- Need to compare results of different programs (Fasta, Blast, Smith-Waterman)
- Need to modify existing scripts to work with new/updated programs and web sites.
- Need to use an existing program's output as input to a different program, not designed for that program:
  - Database search -> Multiple Alignment
  - Multiple Alignment -> Pattern search
  - Need to Organize your data

# Why Learn to Program?

## **Bioinformatics Assembly Analyst**

### Responsibilities:

- Assembling genome sequence data using a variety of tools and parameters and performing the experiments needed to evaluate sequencing strategies
- Using existing software and databases to analyze genomic data and correlating assemblies and sequences with a variety of genetic and physical maps and other biological information
- Identifying problems and serving as point of contact for various groups to propose and implement solutions
- Proposing and implementing upgrades to existing tools and processes to enhance analysis techniques and quality of results
- **Developing and implementing scripts to manipulate, format, parse, analyze, and display genome sequence data; and developing new strategies for analysis and presentation of results.**

### Requirements:

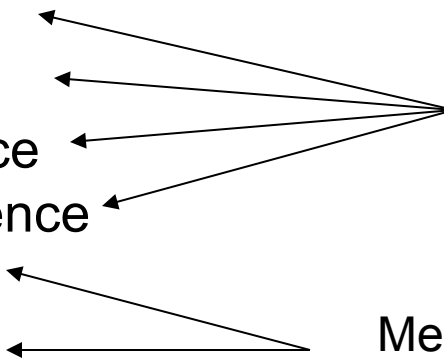
- A bachelor's degree in biology or related field
- At least three years of experience in DNA sequencing and sequence analysis.
- Must possess solid knowledge of sequencing software and public sequencing databases.
- Knowledge of bioinformatics tools helpful.

## Good Languages to Learn In no particular order....

- C/C++
  - Language of choice for most large development projects
- FORTRAN
  - Excellent language for math, not used much anymore
- Java
  - Popular modern object oriented language
- PERL
  - Excellent language for text-processing ([bioperl.org](http://bioperl.org))
- PHP
  - Popular language used to program web interfaces
- Python
  - Language easy to pick up and learn ([biopython.org](http://biopython.org))
- SQL
  - Language used to communicate with a relational database



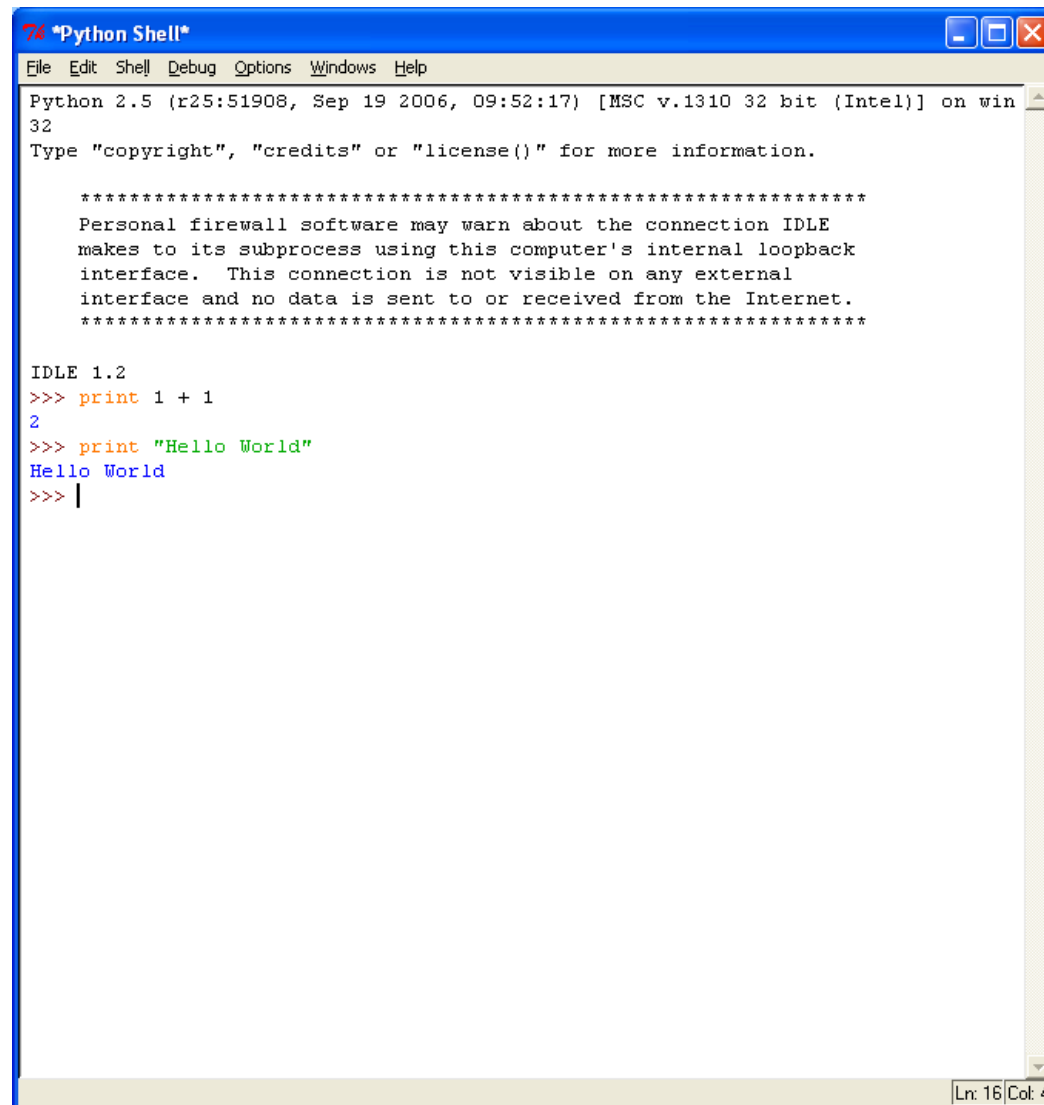
## Python is Object Oriented

- “Object Oriented” is simply a convenient way to organize your data and the functions that operate on that data
  - A biological example of organizing data:
    - Human.CytochromeC.protein.sequence
    - Human.CytochromeC.RNA.sequence
    - Human.CytochromeC.DNA.sequence
  - Some things only make sense in the context that they are used:
    - Human.CytochromeC.DNA.intron
    - Human.CytochromeC.DNA.exon
    - Human.CytochromeC.DNA.sequence
    - Human.CytochromeC.protein.sequence
    - Human.CytochromeC.protein.intron
    - Human.CytochromeC.protein.exon
- Meaningful
- Meaningless
- 

## Downloading and Installing Python

- Go to [www.python.org](http://www.python.org)
- Go to DOWNLOAD section
- Click on [Python 2.6.2 Windows installer](#)
- Save ~10MB file into your hard drive
- Double click on file to install
- Follow instructions
- Start -> All Programs -> Python 2.6 -> Idle

# Idle: The Python Shell



```
*Python Shell*
File Edit Shell Debug Options Windows Help
Python 2.5 (r25:51908, Sep 19 2006, 09:52:17) [MSC v.1310 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.

*****
Personal firewall software may warn about the connection IDLE
makes to its subprocess using this computer's internal loopback
interface. This connection is not visible on any external
interface and no data is sent to or received from the Internet.
*****

IDLE 1.2
>>> print 1 + 1
2
>>> print "Hello World"
Hello World
>>> |
Ln: 16 | Col: 4
```

# Python as a Number Cruncher

```
>>> print 1 + 3
4
>>> print 6 * 7
42
>>> print 6 * 7 + 2
44
>>> print 2 + 6 * 7
44
>>> print 6 - 2 - 3
1
>>> print 6 - ( 2 - 3)
7
>>> print 1 / 3
0
>>>
```

/ and \* higher precedence than + and -

Operators are left associative

Parenthesis can override precedence

integer division truncates fractional part

# Floating Point Expressions

```
>>> print 1.0 / 3.0
```

```
0.333333333333
```

```
>>> print 1.0 + 2
```

```
3.0
```

```
>>> print 3.3 * 4.23
```

```
13.959
```

```
>>> print 3.3e23 * 2
```

```
6.6e+023
```

```
>>> print float(1) / 3
```

```
0.333333333333
```

```
>>>
```

→ 12 decimal digits default precision

→ Mixed operations converted to float

→ Scientific notation allowed

→ Explicit conversion

# String Expressions

<pre>&gt;&gt;&gt; print "aaa" aaa &gt;&gt;&gt; print "aaa" + "ccc" aaaccc &gt;&gt;&gt; len("aaa") 3 &gt;&gt;&gt; len ("aaa" + "ccc") 6 &gt;&gt;&gt; print "aaa" * 4 aaaaaaaaaaaa &gt;&gt;&gt; "aaa" 'aaa' &gt;&gt;&gt; "c" in "atc" True &gt;&gt;&gt; "g" in "atc" False &gt;&gt;&gt;</pre>	<p>→ + concatenates string</p> <p>→ len is a <b>function</b> that returns the length of its <b>argument</b> string</p> <p>→ any expression can be an argument</p> <p>→ * replicates strings</p> <p>→ a <b>value</b> is an <b>expression</b> that yields itself</p> <p>→ in operator finds a string inside another And returns a <b>boolean</b> result</p>
---	---

## Values Can Have (MEANINGFUL) Names

```
>>> numAminoAcids = 20
>>> eValue = 6.022e23
>>> prompt = "Enter a sequence ->"
>>> print numAminoAcids
20
>>> print eValue
6.022e+023
>>> print prompt
Enter a sequence ->
>>> print "prompt"
prompt
>>>
>>> prompt = 5
>>> print prompt
5
>>>
```

→ = **binds** a name to a value

→ use Camel case for compound names

→ prints the value bound to a name

→ = can change the value associated with a name even to a different **type**

# Values Have Types

```
>>> type("hello")
```

```
<type 'str'>
```

```
>>> type(3)
```

```
<type 'int'>
```

```
>>> type(3.0)
```

```
<type 'float'>
```

```
>>> type(eValue)
```

```
<type 'float'>
```

```
>>> type(prompt)
```

```
<type 'int'>
```

```
>>> type(numAminoAcids)
```

```
<type 'float'>
```

```
>>>
```

→ type is another function

→ the “type” is itself a value

→ the type of a name is the type of the value bound to it



## In Bioinformatics Words ...

```
>>> codon="atg"
>>> codon * 3
'atgatgatg'
>>> seq1 ="agcgccttgaattcggcaccaggcaaatctcaaggagaagttccgggggagaaggtgaaga"
>>> seq2 = "cggggagtggggagttgagtcgcaagatgagcgagcggatgtccactatgagcgataata"
>>> seq = seq1 + seq2
>>> seq
'agcgccttgaattcggcaccaggcaaatctcaaggagaagttccgggggagaaggtgaagacggggagtggggagttgagtc
gcaagatgagcgagcggatgtccactatgagcgataata'
>>> seq[1]
'g'
>>> seq[0]
'a'
>>> "a" in seq
True
>>> len(seq1)
60
>>> len(seq)
120
```

→ First nucleotide starts at 0

# More Bioinformatics

## Extracting Information from Sequences

```
>>> seq[0] + seq[1] + seq[2]
'agc'
>>> seq[0:3]
'agc'
>>> seq[3:6]
'gcc'
>>> seq.count('a')
35
>>> seq.count('c')
21
>>> seq.count('g')
44
>>> seq.count('t')
12
>>> long = len(seq)
>>> pctA = seq.count('a')
>>> float(pctA) / long * 100
29.166666666666668
```

Find the first codon from the sequence

get 'slices' from strings:

How many of each base does this sequence contain?

Count the percentage of each base on the sequence.

## Additional Note About Python Strings

```
>>> seq="ACGT"  
>>> print seq  
ACGT
```

```
>>> seq="TATATA"  
>>> print seq  
TATATA
```

Can replace  
one whole  
string with  
another  
whole string

```
>>> seq[0] = seq[1]  
Traceback (most recent call last):  
  File "<pyshell#33>", line 1, in <module>  
    seq[0]=seq[1]  
TypeError: 'str' object does not support item assignment
```

Can **NOT**  
simply replace  
a sequence  
character with  
another  
sequence  
character, but...

```
seq = seq[1] + seq[1:]
```

Can replace a whole string using substrings

# Commenting Your Code!

- How?
  - Precede comment with # sign
  - Interpreter ignores rest of the line
- Why?
  - Make code more readable by others AND yourself?
- When?
  - When code by itself is not evident
    - `# compute the percentage of the hour that has elapsed`
    - `percentage = (minute * 100) / 60`
  - Need to say something but Python cannot express it, such as documenting code changes
    - `percentage = (minute * 100) / 60 # FIX: handle float division`

Please do not over do it



`X = 5 # Assign 5 to x`

# Software Development Cycle

- **Problem Identification**
  - What is the problem that we are solving
- **Algorithm Development**
  - How can we solve the problem in a step-by-step manner?
- **Coding**
  - Place algorithm into a computer language
- **Testing/Debugging**
  - Make sure the code works on data that you already know the answer to
- **Run Program**
  - Use program with data that you do not already know the answer to.

## Lets Try It With Some Examples!

- First, lets learn to **SAVE** our programs in a file:
  - From Python Shell: File -> New Window
  - From New Window: File->Save
- Then, To run the program in the new window:
  - From New Window: Run->Run Module

## Problem Identification

- What is the percentage composition of a nucleic acid sequence
  - DNA sequences have four residues, A, C, G, and T
  - Percentage composition means the percentage of the residues that make up of the sequence

## Algorithm Development

- Print the sequence
- Count characters to determine how many A, C, G and T's make up the sequence
- Divide the individual counts by the length of the sequence and take this result and multiply it by 100 to get the percentage
- Print the results



## Coding

```
seq="ACTGTCGTAT"  
print seq  
Acount= seq.count('A')  
Ccount= seq.count('C')  
Gcount= seq.count('G')  
Tcount= seq.count('T')  
Total = len(seq)  
APct = int((Acount/Total) * 100)  
print 'A percent = %d ' % APct  
CPct = int((Ccount/Total) * 100)  
print 'C percent = %d ' % CPct  
GPct = int((Gcount/Total) * 100)  
print 'G percent = %d ' % GPct  
TPct = int((Tcount/Total) * 100)  
print 'T percent = %d ' % TPct
```

## Let's Test The Program

- First SAVE the program:
  - From New Window: File->Save

## Testing / Debugging

- Six Common Python Coding Errors:
  - Delimiter mismatch: check for matches and proper use.
    - Single and double quotes: ‘ ’ “ ”
    - Parenthesis and brackets: { } [ ] ( )
  - Spelling errors:
    - Among keywords
    - Among variables
    - Among function names
  - Improper indentation
  - Import statement missing
  - Function calling parameters are mismatched
  - Math errors:
    - Automatic type conversion: Integer vs floating point
    - Incorrect order of operations – always use parenthesis.

## Testing / Debugging

```
seq='ACTGTCGTAT"  
print seq;  
Acount= seq.count('A')  
Ccount= seq.count('C')  
Gcount= seq.count('G')  
Tcount= seq.count('T')  
Total = len(seq)  
APct = int((Acount/Total) * 100)  
print 'A percent = %d ' % APct  
CPct = int((Ccount/Total) * 100)  
print 'C percent = %d ' % CPct  
GPct = int((Gcount/Total) * 100)  
print 'G percent = %d ' % GPct  
TPct = int((Tcount/Total) * 100)  
print 'T percent = %d ' % TPct
```

## Let's Test The Program

- First, re-SAVE the program:
  - File->Save
- Then RUN the program:
  - Run->Run Module
- Then LOOK at the Python Shell Window:
  - If successful, the results are displayed
  - If unsuccessful, error messages will be displayed

## Testing/Debugging

- The program says that the composition is:
  - 0%A, 0%C, 0%G, 0%T
- The real answer should be:
  - 20%A, 20%C, 20%G, 40%T
- The problem is in the coding step:
  - Integer math is causing undesired rounding!

## Testing/Debugging

```
seq="ACTGTCGTAT"  
print seq  
Acount= seq.count('A')  
Ccount= seq.count('C')  
Gcount= seq.count('G')  
Tcount= seq.count('T')  
Total = float(len(seq))  
APct = int((Acount/Total) * 100)  
print 'A percent = %d ' % APct  
CPct = int((Ccount/Total) * 100)  
print 'C percent = %d ' % CPct  
GPct = int((Gcount/Total) * 100)  
print 'G percent = %d ' % GPct  
TPct = int((Tcount/Total) * 100)  
print 'T percent = %d ' % TPct
```

Let's change the nucleic acid sequence from  
DNA to RNA...

- If the first line was changed to:
  - seq = "ACUGCUGUAU"
- Would we get the desired result?



## Testing/Debugging

- The program says that the composition is:
  - 20%A, 20%C, 20%G, 0%T
- The real answer should be:
  - 20%A, 20%C, 20%G, 40%U
- The problem is that we have not defined the problem correctly!
  - We designed our code assuming input would be DNA sequences
  - We fed the program RNA sequences

## Problem Identification

- What is the percentage composition of a nucleic acid sequence
  - DNA sequences have four residues, A, C, G, and T
  - In RNA sequences “U” is used in place of “T”
  - Percentage composition means the percentage of the residues that make up of the sequence

## Algorithm Development

- Print the sequence
- Count characters to determine how many A, C, G, T and U's make up the sequence
- Divide the individual A,C,G counts and the sum of T's and U's by the length of the sequence and take this result and multiply it by 100 to get the percentage
- Print the results

## Testing/Debugging

```
seq="ACUGUCGUAU"  
print seq  
Acount= seq.count('A')  
Ccount= seq.count('C')  
Gcount= seq.count('G')  
TUcount= seq.count('T') + seq.count('U')  
Total = float(len(seq))  
APct = int((Acount/Total) * 100)  
print 'A percent = %d ' % APct  
CPct = int((Ccount/Total) * 100)  
print 'C percent = %d ' % CPct  
GPct = int((Gcount/Total) * 100)  
print 'G percent = %d ' % GPct  
TUPct = int((TUcount/Total) * 100)  
print 'T/U percent = %d ' % TUPct
```

## What's Next

- Extend your code to handle the nucleic acid ambiguous sequence characters “N” and “X”
- Extend your code to handle protein sequences