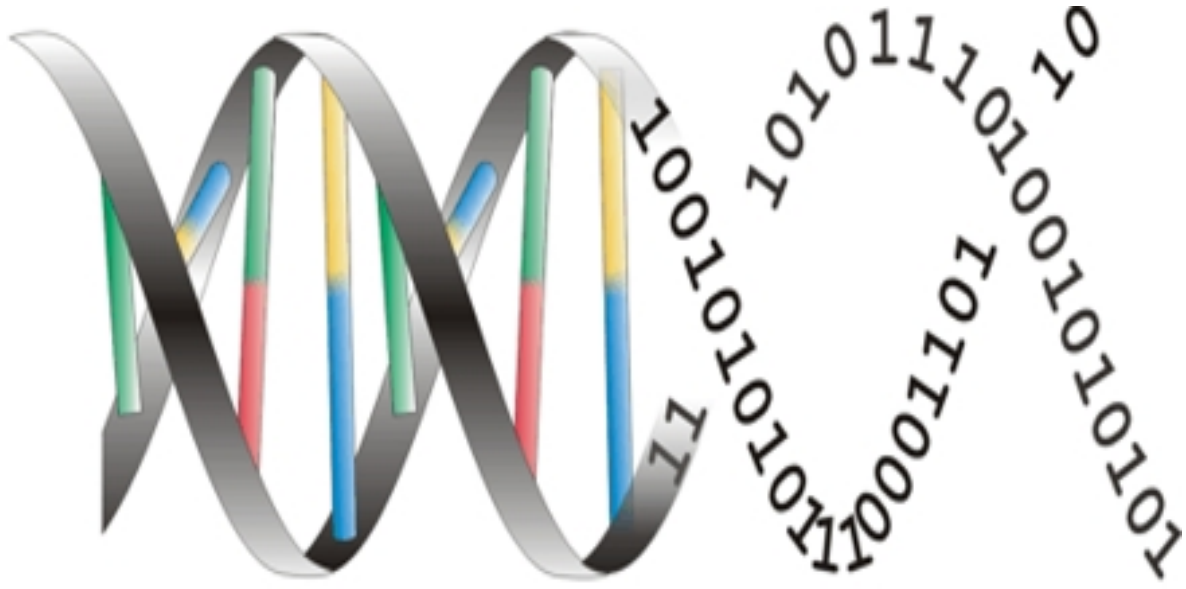
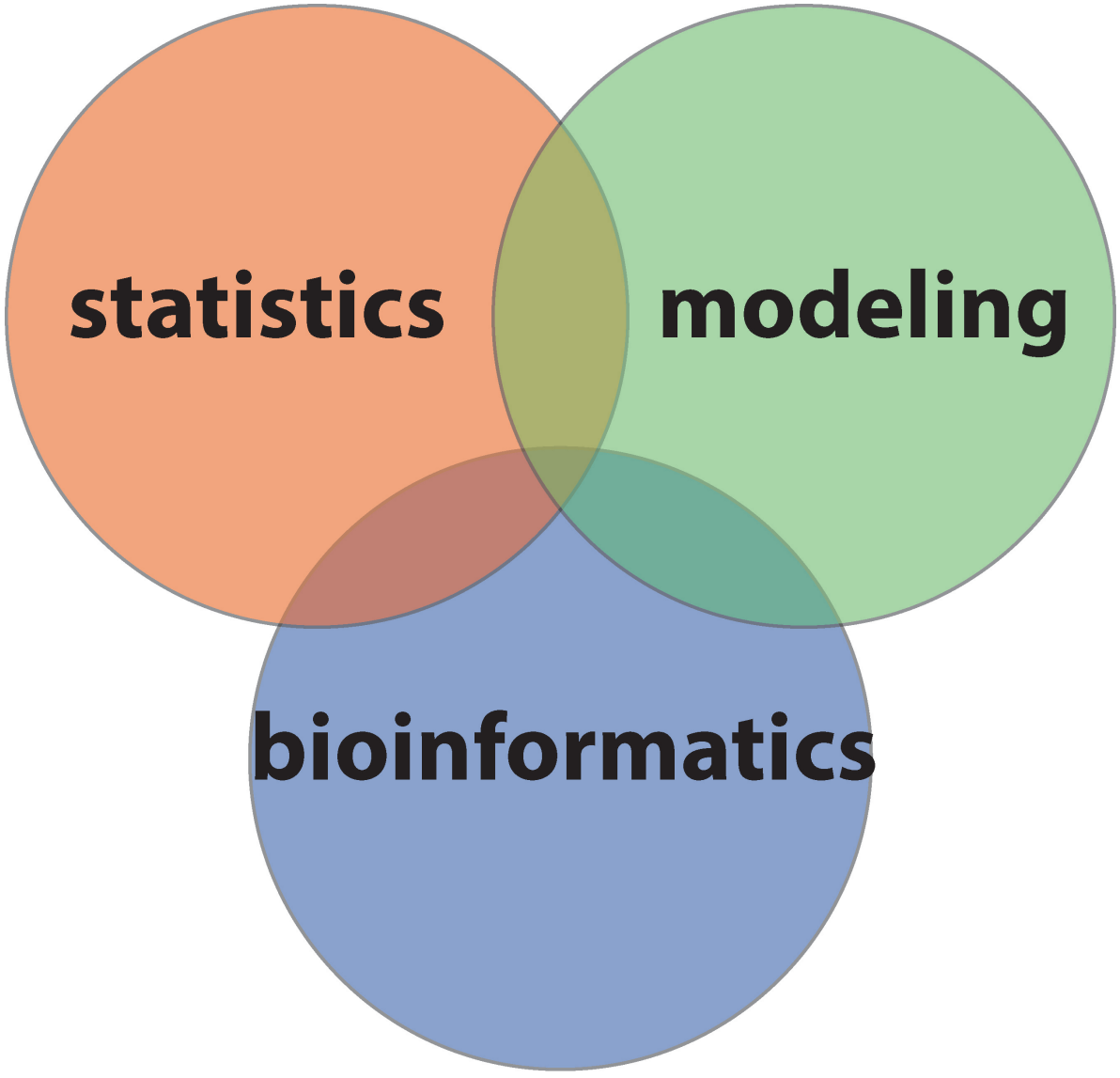
Lecture 01 - Welcome to Introduction to Computational Biology (ICB)

Why biocomputation?



Why biocomputation?



Course Philosophy

- Teach you enough to be dangerous!
- Focus on exposure and general solutions rather than mastery of a particular tool
- Allow you to recognize when to use tools
- Not a bioinformatics, statistics, or modeling course (but those are available)

Learning Goals

1. Use of a powerful means to interact with local and remote computers (Unix)
2. Scripting using Python or R
3. Best practices and applications

We'll spend approximately 1/3 of the semester on each of these goals!

Two principle aspects of programming

1. Conceptual building blocks – the focus of lecture (MW)
2. Language-specific syntax – the focus of tutorials (F)

Class preparation

- No book
- Readings or activities before many class meetings
- Readings and assignments will be available on Sakai
- Materials also on GitHub - `https://github.com/joneslabND/ICB_Fall2017`

Sakai

- Announcements
- Resources
- Forum

Components of the Course

Lots of moving parts because we are covering a lot of ground.

Each of these components are designed and included to enhance your learning!

1. Quizzes
2. Good & Bad
3. Graded Exercises
4. Group Projects
5. Final Exam

Tutorial sections

- doesn't matter what section you are registered for
- what are you more interested in right now - R or Python?
- we'll let you know before Friday where to go, but we won't cover scripting for a few weeks

Cygwin is a program that emulates Unix within Windows.

What's Unix?

- A set of simple, modular tools that work on files organized in a filesystem
- Allows for multitasking and "time sharing"
- Developed in the C programming language at Bell Labs in the 1970's
- Since its development, variations have been created including the open-source Linux and the basis of Mac OSX (Darwin and macOS)
- The "GNUS's Not Unix" (GNU) project was instrumental in expansion of the open-source Lnux and development of a large number of open-source libraries used today

Why Unix?

- Efficient
- Multi-tasking
- Modularity allows for user creation of powerful tools
- Portable and easy for communication with other computers, including large computing resources

How does Unix work?

- A *shell* interprets commands provided by the user and passes them to the *kernel*, which does all of the work in the operating system
- These commands are often built from a set of existing functions, but can also be custom written by users
- A user can interact with the shell from the command line or write shell scripts in order to automate processes
- The majority of the work done by the kernel fits the pattern of reading a file line-by-line, transforming contents of the file in some way, and writing to a new file

The Unix shell

- Lots of versions of Unix shells have been developed over the years
  - *sh* was the original shell developed at Bell Labs
  - *sh* was upgraded in the late 1970's, but then completely rewritten as the Bourne shell
  - The Bourne-Again shell, *bash* is a GNU Project version of the Bourne shell and is very common today
  - Other common shell varieties you might encounter are C shell, *csh*, and TENEX C shell, *tcsh*

Looking ahead

- Friday's tutorial
  - look at first episode of Shell Lesson on Software Carpentry
  - download required files
  - OSX and Linux users are all set; Windows users need to install CygWin
- Next week
  - continue working through Shell Lesson on Software Carpentry
  - quizzes on Monday and Wednesday