software carpentry

DATA CARPENTRY

Lessons and Objectives

Writing Lessons

- Theme so far – start at the end
  - MCQs – misconceptions
  - Faded examples – remove bits from full example

- Writing lessons is the same
  - Know your audience, what you want them to learn
  - Work backwards from that

Writing Learning Objectives

- Single sentence describing what learner will be *able to do* after a lesson
  - To *demonstrate* learning

- Should be **specific** and **verifiable**, with
  - Measurable/verifiable verb – what learner will do
  - Criteria for acceptable performance

- Also need to understand what kind of learning we are aiming for

Bloom’s Taxonomy

Bloom’s Taxonomy	Examples
Knowledge: recalling learned information	’for’ starts a loop in Python
Comprehension: explaining the meaning of information	describing how a for loop works
Application: applying what one knows to novel, concrete situations	writing a for loop to rename files in Unix shell
Analysis: breaking down a whole into component parts and explaining how each part contributes	explaining how loop body, loop variable and collection relate
Synthesis: assembling components to form a new and integrated whole	using a while/repeat until loop works based on for knowledge
Evaluation: using evidence to make judgments about relative merits of ideas and materials	pros of cons of for loops vs. while loops

Our focus!

- Each level represents a deeper understanding and greater ability to apply it

Exercise

Links to Software/Data Carpentry lessons at top of Etherpad.

*Take a minute to **select one learning objective** from one of those lessons, then complete the following steps to **evaluate it and reword it to make it sharper***

1. ***Identify** the learning objective verb.*
2. ***Decide** what type of learning outcome this applies to (i.e. comprehension, application, evaluation).*
3. ***Reword** the learning objective for a different learning outcome (e.g., from application to knowledge based outcome or vice versa).*
4. ***Pair** up to discuss your rewording or help each other with point 2 or 3 if necessary.*
5. ***Share** the original and your re-worded learning objectives in the Etherpad.*

How are Courses Mostly Designed?

1. Someone tells you to teach something you haven't thought about in ten years
2. You start writing slides to explain what you know about the subject
3. After two or three weeks, you make up an assignment based more or less on what you've taught so far
4. You repeat step 3 several times
5. You stay up late to make up a final exam

Reverse Instructional Design (RID)

- Similar to Test-Driven Development (TDD)

1. Identify what is worth learning
   - e.g. via concept maps
2. Decide what constitutes evidence that learning has taken place
   - Create final exam or other summative assessment
3. Design practice work to prepare learners for summative assessment
   - In-class formative assessments
   - Out of class exercises
4. Sort practices in order of increasing complexity
5. Write short episodes to close the gap between what learners know and what they need to know in order to do each one
   - Classroom lesson consists of several such episodes
   - Each episode builds toward quick formative assessment

Keeps teaching focused on its objectives
Ensures learners prepared for final exam

Software Carpentry Lessons – In Practice

- Most commonly used
  - The Unix Shell
  - Version Control with Git
  - Programming with Python
  - Programming with R
  - R for Reproducible Scientific Analysis
- Others
  - Version Control with Mercurial
  - Using Databases with SQL
  - Programming with MATLAB
  - Automation and Make

Software Carpentry Lessons – In Practice

- Most commonly used
  - The Unix Shell
  - Version Control with Git
  - Programming with Python
  - Programming with R
  - R for Reproducible Scientific Analysis
- Others
  - Version Control with Mercurial
  - Using Databases with SQL
  - Programming with MATLAB
  - Automation and Make

Main aims

Teach a few basic concepts that crop up in many areas of computing:

- Path, home directory
- History, tab completion
- head, tail, grep
- Using pipes
- Using loops

Software Carpentry Lessons – In Practice

- Most commonly used
  - The Unix Shell
  - Version Control with Git
  - Programming with Python
  - Programming with R
  - R for Reproducible Scientific Analysis
- Others
  - Version Control with Mercurial
  - Using Databases with SQL
  - Programming with MATLAB
  - Automation and Make

Main aims

To teach:

- How to keep track of work
- How to collaborate with other people online
- Enough about privacy and licensing to make sensible decisions

Software Carpentry Lessons – In Practice

- Most commonly used
  - The Unix Shell
  - Version Control with Git
  - Programming with Python
  - Programming with R
  - R for Reproducible Scientific Analysis
- Others
  - Version Control with Mercurial
  - Using Databases with SQL
  - Programming with MATLAB
  - Automation and Make

Main aims

To teach building modular programs out of small functions that can be

- Read
- Tested
- Re-used

Hard to teach to novices

Focus on mechanics of doing common operations

Data Carpentry Lessons – In Practice

- Domain-specific
- Cover aspects of data relevant to domain
  - Organisation, analysis, manipulation, visualising tabular data

- Current domains include
  - Ecology, Genomics, Geospatial Data
- Others in development and testing
  - Social Science, semester-long Biology course