20.180:Programming

Introductory Unix & Python Tutorial

Welcome to the Unix and Python Tutorial for 20.180! We expect for you all to attend the tutorial sessions. The first homework assignment assumes basic Python programming knowledge, so it is in your best interest to come to these sessions.

Upon completion of this tutorial, you should know how to do the following:

1. **Log in and out of UNIX**
2. **Manipulate files and execute programs in UNIX**
3. **Start and end a Python session**
4. **Use strings, lists and dictionaries**
5. **Write loops (for- and while-loops) and conditional statements (if-else if-then)**
6. **Open, modify and close a file**
7. **Compose and call simple functions**

Python Reference Books

1. Lutz, Mark, and David Ascher. *Learning Python*. Cambridge, MA: O'Reilly, 1999. ISBN: 1565924649.

2. Martelli, Alex. *Python in a Nutshell*. Cambridge, MA: O'Reilly, 2003. ISBN: 0596001886.

3. Martelli, Alex, and David Ascher, eds. *Python Cookbook*. Cambridge, MA: O'Reilly, 2002. ISBN: 0596001673.

4. Lundh, Fredrik. *Python Standard Library*. Cambridge, MA: O'Reilly, 2001. ISBN: 0596000960.

Foundational Engineering Concepts

1. **Abstraction**: Abstraction is a mechanism to reduce and factor out details so that one can focus on few concepts at a time. (From http://en.wikipedia.org/wiki/Abstraction_%28programming%29)
    1. EXAMPLES: Physics to EE (1800s), in synthetic biology: parts (proteins) --> devices (inverter) --> systems (ring oscillator)
2. **Standardization**: Standardization is the process of publicly establishing a technical standard. (From http://en.wikipedia.org/wiki/Standardization)

    1. EXAMPLES: use of the International System of Units (SI) in science, standardization of screw threads and nuts, use of SBML (systems biology markup language), use of FASTA files to carry sequence data
3. **Decomposition**: Decomposition, otherwise known as factoring or decoupling, refers to the process by which a complex problem or system is broken down into parts that are easier to conceive, understand, program, and maintain. (From http://en.wikipedia.org/wiki/Decomposition_%28computer_science%29)
    1. EXAMPLE: break construction of a building into smaller separate tasks to be handled by experts (strucutral engineer, architect, etc)

Basic programming concepts

1. **Data abstraction**: Data abstraction is the enforcement of a clear separation between the abstract properties of a data type and the concrete details of its implementation. (From http://en.wikipedia.org/wiki/Abstraction_%28programming%29#Data_abstraction)
    1. EXAMPLES: lists and dictionaries in Python
2. **Abstraction of functions / reusable code**: Reusability is the likelihood a segment of structured code can be used again to add new functionalities with slight or no modification. Reusable code reduces implementation time, increases the likelihood that prior testing and use has eliminated bugs and localizes code modifications when a change in implementation is required. Subroutines or functions are the simplest form of reuse. A chunk of code is regularly organized using modules or namespaces. The ability to reuse relies on the ability to build larger things from smaller parts, and being able to identify commonalities among those parts. Reusable code can be implemented within the context of the individual, whereas standardization implies a public specification of interface. (From http://en.wikipedia.org/wiki/Reusability)
    1. EXAMPLE: a function "mRNAtoprotein" which converts an mRNA sequence into an amino acid sequence, which can be used as a black box and called without knowing the details of how the function works
3. **Iteration**: Iteration is the repetition of a process. It can be used both as a general term, synonymous with repetition, and to describe a specific form of repetition with a mutable state (for example the counter "i" in a "for" loop). When used in the first sense, recursion is an example of iteration. (From http://en.wikipedia.org/wiki/Iteration)
    1. EXAMPLES: for loops, while loops
4. **Recursion**: Mathematical recursion involves a function calling on itself over and over until reaching an end state. A commonly used example is the function used to calculate the factorial of an integer. (From http://en.wikipedia.org/wiki/Recursion)
    1. EXAMPLES: Fibonacci numbers: $f(n) = f(n-1) + f(n-2)$, factorials
5. **Object orientation**: The idea behind object-oriented programming (OOP) is that a computer program may be seen as composed of a collection of individual units, or objects, that act on each other, as opposed to a traditional view in which a program may be seen as a collection of functions or procedures, or simply as a list

of instructions to the computer. (From http://en.wikipedia.org/wiki/Object-oriented_programming#Formal_definition)

1. EXAMPLES: C++, Java, Python and C#
2. Other languages with object-oriented features: Ada, BASIC, Lisp, Fortran, Pascal
3. **OOP concepts**
    1. **Class**: the unit of definition of data and behavior; a class (for example, Dog) is the basis of modularity and structure in an object-oriented computer program
    2. **Object**: an instance of a class; for example, Spot the Dog
    3. **Inheritance**: a mechanism for creating subclasses; inheritance provides a way to define a (sub)class as a specialization or subtype of a more general class (as Dog is a subclass of Canine). It is intended to help reuse of existing code.
    4. **Abstraction**: the ability of a program to ignore the details of an object's (sub)class and work at a more generic level when appropriate; for example, Spot the Dog may be treated as a Dog much of the time

6. **Scope**: The scope of a variable describes where in a program's text a variable may be used, while extent (or lifetime) describes when in a program's execution a variable has a value. (From http://en.wikipedia.org/wiki/Scope_%28programming%29)
    1. **Local variable**: A variable that is given local scope. Such variables are accessible only from the function or block in which it is declared.
    2. **Global variable**: A variable that does not belong to any subroutine in particular and can therefore can be accessed from any context in a program.
7. **Algorithm**: a finite set of well-defined instructions for accomplishing some task which, given an initial state, will terminate in a corresponding recognizable end-state. Informally, the concept of an algorithm is often illustrated by the example of a recipe, although many algorithms are much more complex; algorithms often have steps that repeat (iterate) or require decisions (such as logic or comparison). (From http://en.wikipedia.org/wiki/Algorithm#Classification_by_design_paradigm)
    1. EXAMPLES: sort algorithm, search algorithm
    2. **Computational complexity theory**: the branch of the theory of computation that studies the resources required during computation to solve a given problem. The most common resources are time (how many steps it takes to solve a problem) and space (how much memory it takes).
    3. **Big O notation**: Big O (standing for "order of") notation is a mathematical notation used to describe the asymptotic behavior of functions. More precisely, it is used to describe an asymptotic upper bound for the magnitude of a function. Big O notation is useful when analyzing algorithms for efficiency. Big O can also be used to describe the error term in an approximation to a mathematical function.(From http://en.wikipedia.org/wiki/Big_O_notation)

1. EXAMPLE: Consider an instance that is n bits long that can be solved in n^2 steps. We say the problem has time complexity $O(n^2)$.

Low-level programming concepts

1. **Control structures**
    1. **Expressions and operators** (+,==,*, etc)
        1. **Expressions**: An expression in a programming language is a combination of values, variables, operators, and functions that are interpreted (evaluated) according to the particular rules of precedence and of association for a particular programming language, which computes and then produces (returns, in a stateful environment) another value. The expression is said to evaluate to that value. As in mathematics, the expression is (or can be said to have) its evaluated value; the expression is a representation of that value. (From http://en.wikipedia.org/wiki/Expression_%28programming%29)
        2. **Operators**: Programming languages generally have a set of operators that are similar to operators in mathematics: they are somehow special functions. In addition to arithmetic operations they often perform boolean operations on truth values and string operations on strings of text. Unlike functions, operators often provide the primitive operations of the language, their name consists of punctuation rather than alphanumeric characters, and they have special infix syntax and irregular parameter passing conventions. (From http://en.wikipedia.org/wiki/Operator_%28programming%29)
    2. **Loops**: A loop is a sequence of statements which is specified once but which may be carried out several times in succession. (From http://en.wikipedia.org/wiki/Control_structure)
        1. **Count-controlled loops**: (For loops) Loops that can be repeated a certain number of times.
        2. **Condition-controlled loops**: (While loops) Loops that can be repeated until some condition changes.
    3. **Conditional statements**: (If-Then clause) Requests to the computer to make an execution choice based on a given condition. (From http://en.wikipedia.org/wiki/Conditional_statement)
    4. **Subroutines**: (functions, methods, procedures, or subprograms) A portion of code within a larger program, which performs a specific task and is relatively independent of the remaining code. A subroutine is often coded so that it can be executed ("called") several times and/or from several places during a single execution of the program, possibly even by itself. (From http://en.wikipedia.org/wiki/Subroutine)
        1. **Functions**: Function and procedure often denote a subprogram that takes parameters and may or may not have a return value. Many make the distinction between "functions", that possess return

values and appear in expressions, versus "procedures", that possess no return values and appear in statements.

2. **Input / output (I/O)**: The collection of interfaces that different functional units (sub-systems) of an information processing system use to communicate with each other, or the signals (information) sent through those interfaces. Inputs are the signals received by the unit, and outputs are the signals sent from it. (From http://en.wikipedia.org/wiki/Input/output)
    1. EXAMPLE: Keyboards and mice are considered input devices of a computer and monitors and printers are considered output devices of a computer.

Programming stategies

1. **Pseudo-code**: Description of a computer programming algorithm that uses the structural conventions of programming languages, but omits detailed subroutines or language-specific syntax. (From http://en.wikipedia.org/wiki/Pseudocode)
2. **Debugging**: Debugging is a methodical process of finding and reducing the number of bugs, or defects, in a computer program or a piece of electronic hardware thus making it behave as expected. Debugging tends to be harder when various subsystems are tightly coupled, as changes in one may cause bugs to emerge in another. (From http://en.wikipedia.org/wiki/Debugging)
3. **Unit testing / modular coding**: A unit test is a procedure used to verify that a particular module of source code is working properly. The idea about unit tests is to write test cases for all functions and methods so that whenever a change causes a regression, it can be quickly identified and fixed. (From http://en.wikipedia.org/wiki/Unit_testing)
4. **Code validation**: how to know your program does not work
5. **Assertions**: A programming language construct that indicates an assumption on which the program is based. Programmers add assertions to the source code as part of the development process. They are intended to simplify debugging and to make potential errors easier to find. Since an assertion failure often indicates a bug, many assertion implementations will print additional information about the source of the problem (such as the filename and line number in the source code or a stack trace). Most implementations will also halt the program's execution immediately. (From http://en.wikipedia.org/wiki/Assert)
    1. Error handling

Tips on programming style

1. Always comment your code. This will allow other people to understand what you have done, and it can provide a reminder to yourself in the future.
2. Use logical and consistent indentation and spacing. This makes one's code more readable and may be required in some programming languages.

Matlab Tutorial

1. We will be providing a Matlab Tutorial during the semester, which will include:
    1. Getting started with MATLAB
    2. Data structures: Matrices, vectors
    3. Matrix manipulation
    4. Function declaration
    5. Data visualization
    6. Numerical solvers