C S 21A: PYTHON FOR PROGRAMMERS

Foothill College Course Outline of Record

Heading	Value
Units:	4.5
Hours:	4 lecture, 2 laboratory per week (72 total per quarter)
Advisory:	One of the following: C S 1A, 2A, 3A or equivalent.
Degree & Credit Status:	Degree-Applicable Credit Course
Foothill GE:	Non-GE
Transferable:	CSU/UC
Grade Type:	Letter Grade (Request for Pass/No Pass)
Repeatability:	Not Repeatable

Student Learning Outcomes

- A successful student will be able to write and debug Python programs which make use of the fundamental control structures and method-building techniques common to all programming languages. Specifically, the student will use data types, input, output, iterative, conditional, and functional components of the language in his or her programs.
- A successful student will be able to use object-oriented programming techniques to design and implement a clear, well-structured Python program. Specifically, the student will use and design classes and objects in his or her programs.

Description

Introduction to the Python language and environment. Covers topics including object oriented programming, elementary data structures, modules, algorithms, recursion, data abstraction, code style, documentation, debugging techniques and testing.

Course Objectives

The student will be able to:
A. Describe the basic elements of the Python language and the Python interpreter and discuss the differences between Python and other modern languages.
B. Analyze and demonstrate the use of lists and tuples in Python.
C. Describe and use Python dictionaries correctly and demonstrate the use of dictionary methods.
D. Define, analyze and code the basic Python conditional and iterative control structures and explain how they can be nested and how exceptions can be used.
E. Design, implement, test, and debug functions and methods that can be used in programs, and demonstrate the way parameters are passed in such functions and methods.
F. Write classes to demonstrate the ideas of encapsulation, inheritance, interfaces and object oriented program design.
G. Explain and demonstrate methods of error handling and Python exceptions.
H. Demonstrate the understanding of &#65533;magic methods&#65533; through use of these in the context of a Python application.
I. Use pre-written modules and learn the techniques necessary for creating modules.
J. Write to and read from files using intermediate file I/O operations in a Python program.
K. Use an existing library to implement a graphical user interface.
L. Design, implement, test, and debug a program that uses each of the following fundamental programming constructs: string processing, numeric computation, simple I/O, arrays and the Python standard library.
M. Solve problems that have origins in a variety of disciplines including math, science, the Internet and business.

Course Content

A. Introduction to Python
1. Structure of a Python program
2. The Python interactive interpreter
3. Basic input and output
4. Comparison of major languages
5. Online Python resources
B. Lists and Tuples
1. Common sequence operations
2. Lists
3. Tuples
C. Dictionaries
1. Purpose
2. Syntax
3. Basic operations and methods
D. Control Structures
1. Selective and conditional statements
2. Loop statements
3. Nesting levels in control statements
4. Handling exceptions
E. Methods and Functional Programming
1. Parameter passing
2. Functional returns
3. Variable scope
4. Lambda functions
5. Recursion
F. Object-Oriented Programming - Classes and Methods
1. Encapsulation of member data and methods
2. Access modifiers and data privacy
3. Encapsulation and properties
4. Instance vs. static members
5. Instance, class and static methods
6. Object composition (the "has a" relationship)
7. Inheritance (the &#65533;is a&#65533; relationship)
8. Interfaces
9. Object oriented design
G. Error Reporting
1. Raising exceptions
2. Catching exceptions
3. Multiple except clauses
4. Catching two exceptions with one block
5. Catching the object
6. Implementing the non-exceptional situation
7. The finally clause
8. Exceptions and functions
H. Magic Methods
1. Constructors
2. Item access
3. Properties
4. Iterators

5. Generators
I. Modules
1. Module use
2. Creating modules
3. Exploring available Python modules
4. The standard library
5. Events and event driven programming using a predefined module
J. Files
1. Opening files
2. Basic file methods
3. Iterating over file contents
4. Accessing remote files with urllib
K. Graphical User Interfaces
1. Comparison of GUI toolkits
2. GUI features
3. Event handling
L. Essential examples and Assignment Areas
1. String/text processing
2. Numeric computation
3. User interaction through the console
4. Lists and tuples
5. Using Python-defined modules
6. Creating and using a programmer-defined class
M. Applications used throughout course in selected areas
1. Math
2. Physics
3. Chemistry
4. Biology
5. Astronomy
6. Business and finance
7. Internet

Lab Content

A. Familiarization with the beginning-level lab environment.
1. Modify and customize the settings of an Integrated Development Environment (IDE).
2. Use the IDE to create a new programming project.
3. Organize projects within an IDE to make submitting labs and switching project environments an orderly process.
4. Gain experience with the steps needed to edit a simple program.
5. Modify IDE settings to produce an industry standard code style.
B. Exploring the different data types using the compiler/IDE.
1. Gain experience in effectively using the IDE to create code with Python numeric and string types.
2. Gain experience in effectively using the IDE to create code with simple data structures.
3. Use the IDE to assist in defining and using compound data types.
4. Solve syntax and logic problems that arise from typical incorrect formulation of data types.
C. Demonstrating user interaction (I/O) through the console and graphical interface.
1. Play the role of user and programmer, alternately, to establish a user-interaction plan for a program.
2. Evaluate and comment on other students' user-interaction plan.
3. Change modes from source code design (editing mode) to end-user interaction (run mode) in the IDE in order to perform Q/A on the program.
4. Fix poor interaction behavior by adjusting source code and rerunning program until a satisfactory result is achieved.
D. Building a program that demonstrates intelligence though a combination of control statements.
1. Become familiar with selection, loop and nesting to imbue a program with correct logic behavior.
2. Use structured programming to make control structures maintainable.
3. Run the program multiple times to verify that its control statements produce the correct behavior or output under any scenario.
4. Fix incorrect logic behavior by adjusting control structures and rerunning program until a satisfactory result is achieved.
E. Incorporating functions and class methods in programming projects.
1. Gain experience in writing a function/method.
2. Use a previously written function or method in a client program.
3. Refine methods/functions by adding or changing their definitions and observe the result.
4. Deduce the impact of a function's or method's design on the programs that invoke it.
F. Building a program around object-oriented techniques.
1. Use previously written classes to instantiate objects in program.
2. Use the IDE to assist in the creation of a programmer-defined class.
3. Demonstrate the correct choice of class members and methods for each class used.
4. Import and use Python modules in conjunction with user written code.
G. Exploring Lists, Tuples and Dictionaries.
1. Understand the proper use of lists, tuples and dictionaries.
2. Incorporate a of lists, tuple or dictionary into a program to facilitate the solution of an assigned problem.
3. Investigate use of sequence operations to shorten and clarify the logic in programs.
4. Use debugging techniques to solve problems that arise during the testing of a program.
H. Devising and utilizing algorithms.
1. Write a program that uses a combination of techniques such as looping, lists, logic and user I/O, all encapsulated in a coherent algorithm.
2. Test the algorithm by running the program multiple times giving it different initial values or inputs.
3. Implement a sorting or simple searching algorithm using arrays.
4. Transcribe an abstract algorithm into a concrete program that is written and tested using the IDE and submitted online for evaluation.
I. Organizing and debugging multi-class projects.
1. Demonstrate the ability to debug programs that contain multiple classes.
2. Distinguish between interface and implementation in projects by creating classes that are independent of I/O modality.
3. Write individual component classes that are independent of client use and can serve several client programs.
4. Incorporate symbolic constants, statics and instance members into classes in a way that demonstrates a mature understanding of object-oriented programming (OOP) in a lab project.
5. Debug a multi-class project to produce a working program.
J. Building a program that uses class inheritance and interfaces to demonstrate how re-use is handled in OOP.
1. Create a project that contains at least one class intended to be used as a base class.
2. Derive (sub-class) one or more classes from the base class or classes.
3. Create and use an interface to solve a given problem.
K. Exceptions and File I/O in Programming.
1. Write a class that has methods which use exception handling to report errors to the client.
2. Write a test client that uses try/catch blocks to detect exceptions.
3. Implement an algorithm or simulation that reads from and/or writes to files rather than the user screen.
4. Demonstrate various ways errors are handled and reported besides exceptions.

Special Facilities and/or Equipment

A. Access to a computer laboratory with Python interpreters.
B. Website or course management system with an assignment posting component (through which all lab assignments are to be submitted) and a forum component (where students can discuss course material and receive help from the instructor). This applies to all sections, including on-campus (i.e., face-to-face) offerings.
C. When taught via Foothill Global Access on the Internet, a fully functional and maintained course management system through which the instructor and students can interact.
D. When taught via Foothill Global Access on the Internet, students must have currently existing e-mail accounts and ongoing access to computers with internet capabilities.

Method(s) of Evaluation

A. Tests and quizzes
B. Written laboratory assignments which include source code, sample runs and documentation
C. Final examination

Method(s) of Instruction

A. Lectures which include motivation for syntax and use of the Python language and OOP concepts, example programs, and analysis of these programs.
B. Online labs (for all sections, including those meeting face-to-face/on campus) consisting of:
1. A programming assignment webpage located on a college-hosted course management system or other department-approved Internet environment. Here, the students will review the specification of each programming assignment and submit their completed lab work.
2. A discussion webpage located on a college-hosted course management system or other department-approved Internet environment. Here, students can request assistance from the instructor and interact publicly with other class members.
C. Detailed review of programming assignments which includes model solutions and specific comments on the student submissions.
D. In person or online discussion which engages students and instructor in an ongoing dialog pertaining to all aspects of designing, implementing and analyzing programs.
E. When course is taught fully online:
1. Instructor-authored lecture materials, handouts, syllabus, assignments, tests, and other relevant course material will be delivered through a college-hosted course management system or other department-approved Internet environment.
2. Additional instructional guidelines for this course are listed in the attached addendum of CS department online practices.

Representative Text(s) and Other Materials

Lutz, Mark. Learning Python. O'Reilly, 2013.

Types and/or Examples of Required Reading, Writing, and Outside of Class Assignments

A. Reading
1. Textbook assigned reading averaging 30 pages per week
2. Reading the supplied handouts and modules averaging 10 pages per week
3. Reading online resources as directed by instructor though links pertinent to programming
4. Reading library and reference material directed by instructor through course handouts
B. Writing
1. Writing technical prose documentation that supports and describes the programs that are submitted for grades

Discipline(s)

Computer Science