



Department of Mathematics, Computer & Information Science

SUMMER 2019

INTRODUCTION TO SCIENTIFIC PROGRAMMING CS 2521

Instructor: Frank Sanacory

Email: sanacoryf@oldwestbury.edu

Office: NAB 2014

Website: sanacory.net

TEXTBOOKS: A Primer on Scientific Programming with Python (Texts in Computational Science and Engineering) 4th ed.

2014 Edition by Hans Petter Langtangen. ISBN-13:978-3642549588.

PREREQUISITE: MA 2310: Calculus I.

COURSE DESCRIPTION: A fast paced introduction to program design and analysis: algorithmic processes, basic programming techniques, program specification & structure, program development, debugging and testing. Emphasis on scientific applications. Discussion of storage classes, files and string manipulation. Basic data structures and algorithms, data abstractions and object-oriented programming. Students learn the language features of Python with an eye toward scientific and data analysis applications.

COURSE OBJECTIVES: At the completion of this course students will

- Understand what is a program a computer hardware software compiler and basic computer operation.
- Understand various data containers and control structures.
- Be able to access files, write to files and to and from other data streams.
- Understand the object model.
- Be able to use abstract data structures and implement algorithms.
- Apply mathematical and scientific algorithms to various scientific scenarios.

COURSE EVALUATION & GRADING:

Midterm	30%
Final	30%
Lab/Programs	30%
Participation and daily quizzes	10%

A	93-100	B+	87-89	C+	77-79	D+	67-69		
A-	90-92	B	83-86	C	73-76	D	63-66	F	0-69
		B-	80-82	C-	70-72	D-	60-62		

ACADEMIC INTEGRITY POLICY

As members of the Old Westbury community, students are expected to adhere to standards of honesty and ethical behavior. Plagiarism and other types of academic dishonesty are condemned at all academic institutions. These acts detract from the student's intellectual and personal growth by undermining the processes of higher learning and the struggle with one's own expression of ideas and information.

Good academic procedure requires giving proper credit when using the words or ideas of others.

Plagiarizing means “presenting somebody else’s words or ideas without acknowledging where those words and ideas come from” (Ann Raimés, *Keys for Writers*, 7th ed., p.135). Examples include:

- copying material from the Internet or other sources and presenting it as one’s own
- using any author’s words without quotation marks; using any quotation without credit
- changing any author’s words slightly and presenting them as one’s own
- using ideas from any source (even in one’s own words) without proper credit
- turning in any assignment containing material written by someone else (including tutor or friend); buying work and submitting it as one’s own
- submitting the same assignment in more than one class without permission of the instructor

Know what plagiarism is and how to avoid it; for guidance see Raimés or any other college writing handbook.

Other types of academic dishonesty include unauthorized collaboration or copying of students’ work (cheating); falsifying grades or other assessment measures; destroying the academic work of another student; the dishonest use of electronic devices; and others. When detected and verified, plagiarism and other academic dishonesty will have serious consequences.

Please note: In this matter, ignorance of the Academic Integrity Policy is never an acceptable excuse.

ACCOMMODATIONS FOR STUDENTS WITH DISABILITIES:

If you have or suspect you may have a physical, psychological, medical or learning disability that may impact your course work, please contact Stacey DeFelice, Director, The Office of Services for Students with Disabilities (OSSD), NAB, 2065, Phone: 516-628-5666, Fax (516) 876-3005, TTD: (516) 876-3083. E-mail: defelices@oldwestbury.edu. The office will help you determine if you qualify for accommodations and assist you with the process of accessing them. All support services are free and all contacts with the OSSD are strictly confidential. SUNY/Old Westbury is committed to assuring that all students have equal access to all learning activities and to social activities on campus.

Topics to be covered

Chapter 1 Computing with formulas

Chapter 2 Loops and lists

Chapter 3 Functions and branching

Chapter 4 User input and error handling

Chapter 5 Array computing and curve plotting

Chapter 6 Dictionaries and Strings

Chapter 7 Introduction to classes

Chapter 8 Random numbers and simple games

Chapter 9 Object-oriented programming

Some applications may include

- Sequences and difference equations
- Introduction to discrete calculus
- Introduction to differential equations
- Probability and Statistics (Uniform and Normal Distributions)
- Monte Carlo Simulations

Table of Contents for A Primer on Scientific Programming with Python.

1	Computing with formulas	
1.1	The first programming encounter: a formula	
1.1.1	Using a program as a calculator	
1.1.2	About programs and programming	
1.1.3	Tools for writing programs	
1.1.4	Writing and running your first Python program	
1.1.5	Warning about typing program text	
1.1.6	Verifying the result	
1.1.7	Using variables	
1.1.8	Names of variables	
1.1.9	Reserved words in Python	
1.1.10	Comments	
1.1.11	Formatting text and numbers	
1.2	Computer science glossary	
1.3	Another formula: Celsius-Fahrenheit conversion	
1.3.1	Potential error: integer division	
1.3.2	Objects in Python	
1.3.3	Avoiding integer division	
1.3.4	Arithmetic operators and precedence	
1.4	Evaluating standard mathematical functions	
1.4.1	Example: Using the square root function	
1.4.2	Example: Computing with $\sinh x$	
1.4.3	A first glimpse of round-off errors	
1.5	Interactive computing	
1.5.1	Using the Python shell	
1.5.2	Type conversion	
1.5.3	IPython	
1.6	Complex numbers	
1.6.1	Complex arithmetics in Python	
1.6.2	Complex functions in Python	
1.6.3	Unified treatment of complex and real functions	
1.7	Symbolic computing	
1.7.1	Basic differentiation and integration	
1.7.2	Equation solving and Taylor series	
2.	Loops and lists	
2.1	While loops	
2.1.1	A naive solution	
2.1.2	While loops	
2.1.3	Boolean expressions	
2.1.4	Loop implementation of a sum	
2.2	Lists	
2.2.1	Basic list operations	
2.2.2	For loops	
2.3	Alternative implementations with lists and loops	
2.3.1	While loop implementation of a for loop	
2.3.2	The range construction	
2.3.3	For loops with list indices	
2.3.4	Changing list elements	
2.3.5	List comprehension	
2.3.6	Traversing multiple lists simultaneously	
2.4	Nested lists	
2.4.1	A table as a list of rows or columns	

2.4.2	Printing objects	
2.4.3	Extracting sublists	
2.4.4	Traversing nested lists	
2.5	Tuples	
2.6	Summary	
2.6.1	Chapter topics	
2.6.2	Example: Analyzing list data	
2.6.3	How to find more Python information	
3	Functions and branching	
3.1	Functions	
3.1.1	Mathematical functions as Python functions	
3.1.2	Understanding the program flow	
3.1.3	Local and global variables	
3.1.4	Multiple arguments	
3.1.5	Function argument of global variable?	
3.1.6	Beyond mathematical functions	
3.1.7	Multiple return values	
3.1.8	Computing sums	
3.1.9	Functions with no return values	
3.1.10	Keyword arguments	
3.1.11	Doc strings	
3.1.12	Functions as arguments to functions	
3.1.13	The main program	
3.1.14	Lambda functions	
3.2	Branching	
3.2.1	If-else blocks	
3.2.2	Inline if tests	
3.3	Mixing loops, branching, and functions in bioinformatics	
3.3.1	Counting letters in DNA strings	
3.3.2	Efficiency assessment	
3.3.3	Verifying the implementations	
4	User input and error handling	
4.1	Asking questions and reading answers	
4.1.1	Reading keyboard input	
4.2	Reading from the command line	
4.2.1	Providing input on the command line	
4.2.2	A variable number of command-line arguments	..
4.2.3	More on command-line arguments	
4.3	Turning user text into live objects	
4.3.1	The magic eval function	
4.3.2	The magic exec function	
4.3.3	Turning string expressions into functions	
4.4	Option-value pairs on the command line	
4.4.1	Basic usage of the argparse module	
4.4.2	Mathematical expressions as values	
4.5	Reading data from file	
4.5.1	Reading a file line by line	
4.5.2	Alternative ways of reading a file	
4.5.3	Reading a mixture of text and numbers	
4.6	Writing data to file	
4.6.1	Example: Writing a table to file	
4.6.2	Standard input and output as file objects	
4.6.3	What is a file, really?	
4.7	Handling errors	

4.7.1 Exception handling	
4.7.2 Raising exceptions	
4.8 A glimpse of graphical user interfaces	
4.9 Making modules	
4.9.1 Example: Interest on bank deposits	
4.9.2 Collecting functions in a module file	
4.9.3 Test block	
4.9.4 Verification of the module code	
4.9.5 Getting input data	
4.9.6 Doc strings in modules	
4.9.7 Using modules	
4.9.8 Distributing modules	
4.9.9 Making software available on the Internet	
5 Array computing and curve plotting	
5.1 Vectors	
5.1.1 The vector concept	
5.1.2 Mathematical operations on vectors	
5.1.3 Vector arithmetics and vector functions	
5.2 Arrays in Python programs	
5.2.1 Using lists for collecting function data	
5.2.2 Basics of numerical Python arrays	
5.2.3 Computing coordinates and function values	
5.2.4 Vectorization	
5.3 Curve plotting	
5.3.1 Matplotlib; pylab	
5.3.2 Matplotlib; pyplot	
5.3.3 SciTools and Easyviz	
5.3.4 Making animations	
5.3.5 Making videos	
5.3.6 Curve plots in pure text	
5.4 Plotting difficulties	
5.4.1 Piecewise defined functions	
5.4.2 Rapidly varying functions	
5.5 More advanced vectorization of functions	
5.5.1 Vectorization of StringFunction objects	
5.5.2 Vectorization of the Heaviside function	
5.5.3 Vectorization of a hat function	
5.6 More on numerical Python arrays	
5.6.1 Copying arrays	
5.6.2 In-place arithmetics	
5.6.3 Allocating arrays	
5.6.4 Generalized indexing	
5.6.5 Testing for the array type	
5.6.6 Compact syntax for array generation	
5.6.7 Shape manipulation	
5.7 Higher-dimensional arrays	
5.7.1 Matrices and arrays	
5.7.2 Two-dimensional numerical Python arrays	
5.7.3 Array computing	
5.7.4 Two-dimensional arrays and functions of two variables	
5.7.5 Matrix objects	
6 Dictionaries and Strings	
6.1 Dictionaries	
6.1.1 Making dictionaries	

6.1.2 Dictionary operations	
6.1.3 Example: Polynomials as dictionaries	
6.1.4 Dictionaries with default values and ordering	
6.1.5 Example: File data in dictionaries	
6.1.6 Example: File data in nested dictionaries	
6.1.7 Example: Reading and plotting data recorded at specific dates	
6.2 Strings	
6.2.1 Common operations on strings	
6.2.2 Example: Reading pairs of numbers	
6.2.3 Example: Reading coordinates	
6.3 Reading data from web pages	
6.3.1 About web pages	
6.3.2 How to access web pages in programs	
6.3.3 Example: Reading pure text files	
6.3.4 Example: Extracting data from HTML	
6.3.5 Handling non-English text	
6.4 Reading and writing spreadsheet files	
6.4.1 CSV files	
6.4.2 Reading CSV files	
6.4.3 Processing spreadsheet data	
6.4.4 Writing CSV files	
6.4.5 Representing number cells with Numerical Python arrays	
6.4.6 Using more high-level Numerical Python functionality	
6.5 Examples from analyzing DNA	
6.5.1 Computing frequencies	
6.5.2 Analyzing the frequency matrix	
6.5.3 Finding base frequencies	
6.5.4 Translating genes into proteins	
6.5.5 Some humans can drink milk, while others cannot	
7. Introduction to classes	
7.1 Simple function classes	
7.1.1 Challenge: functions with parameters	
7.1.2 Representing a function as a class	
7.1.3 Another function class example	
7.1.4 Alternative function class implementations	
7.1.5 Making classes without the class construct	
7.2 More examples on classes	
7.2.1 Bank accounts	
7.2.2 Phone book	
7.2.3 A circle	
7.3 Special methods	
7.3.1 The call special method	
7.3.2 Example: Automagic differentiation	
7.3.3 Example: Automagic integration	
7.3.4 Turning an instance into a string	
7.3.5 Example: Phone book with special methods	
7.3.6 Adding objects	
7.3.7 Example: Class for polynomials	
7.3.8 Arithmetic operations and other special methods	
7.3.9 Special methods for string conversion	
7.4 Example: Class for vectors in the plane	
7.4.1 Some mathematical operations on vectors	
7.4.2 Implementation	
7.4.3 Usage	
7.5 Example: Class for complex numbers	
7.5.1 Implementation	

7.5.2	Illegal operations	
7.5.3	Mixing complex and real numbers	
7.5.4	Dynamic, static, strong, weak, and duck typing . .	
7.5.5	Special methods for “right” operands	
7.5.6	Inspecting instances	
7.6	Static methods and attributes	
8.	Random numbers and simple games	
8.1	Drawing random numbers	
8.1.1	The seed	
8.1.2	Uniformly distributed random numbers	
8.1.3	Visualizing the distribution	
8.1.4	Vectorized drawing of random numbers	
8.1.5	Computing the mean and standard deviation	
8.1.6	The Gaussian or normal distribution	
8.2	Drawing integers	
8.2.1	Random integer functions	
8.2.2	Example: Throwing a die	
8.2.3	Drawing a random element from a list	
8.2.4	Example: Drawing cards from a deck	
8.2.5	Example: Class implementation of a deck	
8.3	Computing probabilities	
8.3.1	Principles of Monte Carlo simulation	
8.3.2	Example: Throwing dice	
8.3.3	Example: Drawing balls from a hat	
8.3.4	Random mutations of genes	
8.3.5	Example: Policies for limiting population growth .	
8.4	Simple games	
8.4.1	Guessing a number	
8.4.2	Rolling two dice	
8.5	Monte Carlo integration	
8.5.1	Derivation of Monte Carlo integration	
8.5.2	Implementation of standard Monte Carlo integration	
8.5.3	Area computing by throwing random points	
8.6	Random walk in one space dimension	
8.6.1	Basic implementation	
8.6.2	Visualization	
8.6.3	Random walk as a difference equation	
8.6.4	Computing statistics of the particle positions	
8.6.5	Vectorized implementation	
8.7	Random walk in two space dimensions	
8.7.1	Basic implementation	
8.7.2	Vectorized implementation	
9.	Object-oriented programming	
9.1	Inheritance and class hierarchies	
9.1.1	A class for straight lines	
9.1.2	A first try on a class for parabolas	
9.1.3	A class for parabolas using inheritance	
9.1.4	Checking the class type	
9.1.5	Attribute vs inheritance: has-a vs is-a relationship	
9.1.6	Superclass for defining an interface	
9.2	Class hierarchy for numerical differentiation	
9.2.1	Classes for differentiation	
9.2.2	Verification	
9.2.3	A flexible main program	
9.2.4	Extensions	

9.2.5 Alternative implementation via functions	
9.2.6 Alternative implementation via functional programming	
9.2.7 Alternative implementation via a single class	
9.3 Class hierarchy for numerical integration	
9.3.1 Numerical integration methods	
9.3.2 Classes for integration	
9.3.3 Verification	
9.3.4 Using the class hierarchy	
9.3.5 About object-oriented programming	
9.4 Class hierarchy for making drawings	
9.4.1 Using the object collection	
9.4.2 Example of classes for geometric objects	
9.4.3 Adding functionality via recursion	
9.4.4 Scaling, translating, and rotating a figure	
9.5 Classes for DNA analysis	
9.5.1 Class for regions	
9.5.2 Class for genes	
9.5.3 Subclasses	