COMP 141 — Computer Science I: Programming Fundamentals — Spring 2018
CRN 28278

Instructor:	Phillip Kirlin
Meetings:	MWF 9–9:50am, Briggs 019
Course website:	`http://www.cs.rhodes.edu/141kirlin`.
Email:	`kirlinp@rhodes.edu` (please include "CS 141" somewhere in the subject)
Office hours:	See website for scheduled office hours; I am also available by appointment.

Official Course Description: An introduction to the fundamental concepts and practices of procedural programming. Topics include data types, control structures, functions, arrays, files, and the mechanics of running, testing, and debugging. Emphasis is placed on program design and problem solving techniques. The course also includes an introduction to the historical and social context of computing and an overview of computer science as a discipline.

Unofficial Course Description: CS 141 is a required course for computer science majors and should be taken during the first year. It is the first course in the sequence for majors and offers an introduction to the fundamental principles of programming, abstraction, and design.

This course is aimed at helping students acquire the reasoning and abstraction skills needed to design algorithms and implement them as computer programs, while also illustrating some of the "big ideas" of computer science. This course teaches one how to think as a computer scientist, by teaching the process of building abstractions to hide implementation details, and of controlling the intellectual complexity of designing large software systems by decomposing problems into simpler sub-problems. We will explore the joy and beauty of computing and see how computing skills are applicable in everyday life.

Though computer science is more than just programming, knowing how to translate your thoughts into code is an important skill for a computer scientist to have. This course will use the Python programming language as the vehicle for exploration of fundamental computer science concepts. However, this is not a course about Python; it is about the structure and interpretation of computer programs.

Course Objectives: At the end of this course, you should be able to

- create algorithms to solve simple computational problems;
- use the Python programming language to implement, test, and debug algorithms for solving simple problems;
- apply the techniques of abstraction and decomposition to break a problem into smaller, easily-understandable pieces;
- understand and modify algorithms and programs written by someone else; and
- apply consistent documentation and style standards to the programs you write.

Text: *Programming in Python 3.3* (online textbook by zyBooks).

Supplemental materials may be distributed in class. The textbook is designed to serve as a reference and there may be material discussed in class that the textbook does not cover.

Prerequisites: None. This course does not assume any previous programming or computer science experience. You are expected to have a reasonable high-school mathematics background to appreciate the use of the mathematical notation.

Coursework:

	Tentative weight	Tentative date
Programming projects	30%	
zyBook assignments	10%	
Midterm 1	18%	Wednesday, February 14, in class
Midterm 2	18%	Wednesday, April 4, in class
Comprehensive final exam	24%	Saturday, May 5, 8:30am

Grades of A–, B–, C–, and D– are guaranteed with final course grades of 90%, 80%, 70%, and 60%, respectively. If your final course grade falls near a letter grade boundary, I may take into account participation, attendance, and/or improvement during the semester.

Office Hours: In addition to regular office hours, am also available immediately after class for short questions. You never need an appointment to see me during regular office hours; you can just come by. Outside of regular office hours, feel free to stop by my office, and if I have time, I'll try to help you. If I don't have time at that moment, we'll set up an appointment for a different time. Don't be shy about coming by my office or sending me email if you can't make my regular office hours. I always set aside time each week for "unscheduled" office hours.

Attendance: Attendance is expected for each class. If your attendance deteriorates, you will be referred to the dean and asked to drop the course. Attendance, participation, and apparent overall improvement trend may be considered in assigning a final grade. Attendance will be checked each class lecture period. After five unexcused absences, each additional absence will reduce the final grade for the course by one letter grade.

Workload: It is important to stay current with the material. Learning to program takes regular practice, and you should be prepared to devote at least 2–3 hours outside of class for each in-class lecture. In particular, you should expect to spend a significant amount of time for this course working on a computer trying example programs and developing programming assignments. Do not wait to the last minute to start your programming assignments.

Programming Assignments:

- All programs assigned in this course must be written in Python. The particular Python environment that will be used in this course is available in the computer labs on campus. You can also download the software onto your own computer from http://www.python.org/download. Make sure to get the latest version of Python 3 (probably 3.6.4). Your programs must run correctly, however, on the lab computers.
- Back up your code somewhere as you're working on your assignments. Computer crashes or internet downtime are not valid excuses for missing a deadline.
- Programming assignments will be either done using the online zyBook environment or turned in via Moodle. Emailed programs will not be accepted. In general, I do not accept late assignments.
- You are allowed to use the course textbook and the course notes for these programs. The use of any other material is forbidden.
- Grades are assigned to programs as follows by this general guideline:

- A (100 pts): The program is carefully designed, efficiently implemented, well documented, and produces clearly formatted, correct output.
- A- (93 pts): The program is an 'A' program with one or two of the minor problems described for grade 'B.'
- B (85 pts): The program typically could easily have been an A program, but it may have minor/careless problems such as poor, inadequate, or incomplete documentation; several literal values where symbolic constants would have been appropriate; wrong file names (these will be specified per program assignment); sloppy code format; minor efficiency problems; etc. (This is not an exhaustive list.) You would be wise to consider B the default grade for a working program — this might encourage you to review and polish your first working draft of an assignment to produce a more professional quality final version of your program.
- C (75 pts): The program has more serious problems: incorrect output or crashes for important special cases (the "empty" case, the "maxed-out" case, etc.), failure to carefully follow design and implementation requirements spelled out in the assignment, very poor or inefficient design or implementation, near complete absence of documentation, etc.
- D (60 pts): The program runs, but it produces clearly incorrect output or crashes for typical cases. Or, it may deviate greatly from the design or implementation requirements stated in the assignment description.
- F (35 pts): Typically, an 'F' program produces no correct output, or it may not even run. It may "look like a program" when printed as a hard copy, but there remains much work to be done for it to be a correct, working program.

Coding Style: Desiging algorithms and writing the corresponding code is not a dry mechanical process, but an art form. Well-written code has an aesthetic appeal while poor form can make other programmers (and instructors) cringe. Programming assignments will be graded based on correctness and style. To receive full credit for graded programs, you must adhere to good programming practices. Therefore, your assignment must contain the following:

- A comment at the top of the program that includes the author of the program, the date or dates, and a brief description of what the program does
- Concise comments that summarize major sections of your code, along with a comment for each function in your code that describes what the function does (unless it is under three lines of code and the purpose is obvious from the name of the function).
- Meaningful variable and function names
- Well-organized code
- White space or comments to improve legibility
- Avoidance of large blocks of copy-and-pasted code

Class Conduct:

- I encourage everyone to participate in class. Raise your hand if you have a question or comment. Please don't be shy about this; if you are confused about something, it is likely that someone else is confused as well. Teaching and learning is a partnership between the instructor and the students, and asking questions not only helps you understand the material, it also helps me know what I'm doing right or wrong.

- Do not use your cell phone while in class, and keep the ringer on silent.
- If you cannot make it to class for whatever reason, make sure that you know what happened during the lecture that you missed. It is your responsibility, and nobody else's, to do so. The best way to do this is to ask a classmate.
- If you have to leave a class early, inform the instructor in advance. It is rude to walk out in the middle of a lecture.

Students With Disabilities: If you have a documented disability and wish to receive academic accommodations, please contact the Office of Student Disability Services at x3885 as soon as possible.

Collaboration: All programming assignments done outside of class, both the major Python assignments and the zyBook exercises, should be completed on one's own. Occasionally there will be collaborative assignments completed during class, but all homework assignments must be completed individually. Put very simply, do not discuss any programming assignments or share code with anybody else. I will be checking for similar assignments and will turn violations over to the honor council.

Academic Integrity: Plagiarism, cheating, and similar anti-intellectual behavior are serious violations of academic ethics and will be correspondingly penalized. If you are concerned about a possible violation of this kind, please talk with me. I understand that being a student at Rhodes can be stressful sometimes and you will have many demands on your time. However, I would much rather have you turn in a partially-completed assignment or do poorly on a test than have you violate the Rhodes Honor Code. I can — and very much want to — help you if you don't understand the material, but violations of academic integrity will be dealt with harshly.

Unless otherwise specified, everything you submit in this course must be your own work and represent your individual effort. These are all included in the definition of reportable Honor Code violations for this course: copying all or part of a solution to a problem, downloading a solution from the internet and submitting it as your own, having someone else provide the solution for you, or allowing someone else to copy from you. If you have any doubt about what type of behavior is acceptable, please talk with me.

Diversity: A diverse learning community is a necessary element of a liberal arts education, for self-understanding is dependent upon the understanding of others. We are committed to fostering a community in which diversity is valued and welcomed. To that end any discrimination or harassment on the basis of race, gender, color, age, religion, disability, sexual orientation, gender identity or expression, genetic information, and national or ethnic origin, will not be tolerated in the classroom.

We are committed to providing an open learning environment. Freedom of thought, a civil exchange of ideas, and an appreciation of diverse perspectives are fundamental characteristics of a community that is committed to critical inquiry. To promote such an academic and social environment we expect integrity and honesty in our relationships with each other and openness to learning about and experiencing cultural diversity. We believe that these qualities are crucial to fostering social and intellectual maturity and personal growth.

Intellectual maturity also requires individual struggle with unfamiliar ideas. We recognize that our views and convictions will be challenged, and we expect this challenge to take place in a climate of open-mindedness and mutual respect.

Course Topics: (not necessarily in this order)

- Algorithm design, including the concepts of abstraction and algorithmic decomposition
- Testing and debugging programs
- Data types, variables, constants, literals, operators, expressions, and statements
- Conditional and iterative control flow structures
- Functions
- Strings
- One- and two-dimensional lists
- File reading and processing