COSC 1010 Fall 2019 — Syllabus

Welcome to COSC 1010, the beginning programming class in Python at Marquette University. This class intends to teach you the basics of problem solving using a modern programming language that incidentally is the favorite language of data science (together with R, but ahead of Java). The intends to teach you how to express solutions programmatically, to find and fix errors, and to introduce you to the world of Graphical User Interfaces, and the imperative programming, functional programming, and object oriented programming styles.

Course Objectives

After successfully completing this class, students will be capable to

- Solve problems by abstraction and by breaking them into smaller parts
- Express solutions programmatically
- Understand and competently use basic programming tools such as conditional statements, loops, functions, classes, and objects
- Think programmatically at a beginner's level
- Program fluently in Python
- Design and plan a small software piece
- Program in the imperative and object-oriented style of programming.

Learning Outcomes:

After successfully completing the class, students will possess

- the capability to express an algorithm in Python using control structures and built-in data structures efficiently and correctly
  - using the imperative style of programming
  - using the object oriented style of programming
- the capability to decompose a more complex task into simpler subtasks and implement these subtasks in individual functions or methods
- the capability to interact programmatically with files and directories
- an understanding of event-based programming in a GUI application with TkInter
- the capability to decompose simple tasks (such as processing a file with numerical data and derive statistical means, developing a simple desk calculator, writing a numerical integrator for functions) into individual processing steps and modules
- an appreciation of online and offline resources for successfully performing programming tasks and the capability to use efficiently a subset of them
- the capability to use comments and docstrings in an appropriate manner
- an appreciation for the difficulty of working in small teams

Text Book

Allen Downey, Think Python 2e: How to think like a computer scientist, greenteapress.com. This book is freely available as an e-book but can also be ordered as a hardcopy published by O'Reilly. The book covers most, but not all contents of the class. There are many other resources including complete Python courses on the Web.

Contents

- Examples of algorithms; arithmetic operations, values and types, interactive and programmatic python.

- Variables, expressions and statements.
- Functions, function calls, function definitions.

- Conditionals and recursions.
- Iterations
- Built-in data structures
  - Strings
  - Lists
  - Dictionaries
  - Tuples
- Interaction with the file system
- Elements of functional programming: List and dictionary comprehensions
- Object oriented design
  (a) Programmer defined types
  (b) Classes and functions
  (c) Classes and methods
  (d) Overloading methods and operator
- Exceptions
- Introduction to graphic programming with Tkinter

Course Methodology

This is an **inverted** class with a plethora of course materials located at tschwarz.mscs.mu.edu (under Classes). Students are expected to prepare for each class by:

- Watching the presentation
- Check progress according to the "achievement sheet"
- Doing the self-test exercises
- Preparing the "individual quiz"

The preparation should take between 30 - 60 minutes per class for 90% of the student body. If you do not prepare yourself for a class, you are wasting the class time and will have to work much harder to make it up. If according to the individual quizzes, a student repeatedly fails to prepare for class, then we will have to take that individual out of the group-based activities and that student will have to work on their own.

At the beginning of each class, we will spend 2-3 minutes to fill in an individual and a group quiz. You will be assigned to a group according to programming experience, but we will change groups during the semester based on performance. The individual quiz is to be solved individually and covers material from the web-presentation. Its contents will be known to you. The group quiz encourages you to argue with peers and also measures your understanding from the day's web-presentation.

You will then program in pairs solving the problems in the day's activities. We will guide you through the programming tasks, but also encourage you to use outside sources such as the many Python tutorials on the web, Python manual pages, the book, or websites such as stackoverflow. The activities sheet contains enough problems to keep a programmer with some previous experience occupied for an hour, but you are not expected to solve more than about half of the problems. We do not want good students to be bored, but we are more interested in students who are new at this. Therefore, we make sure that the good students are occupied with becoming better, while the normal students become good. If you are an experienced Python programmer we will find ways to occupy you productively in class, but you will not be excused a class.

If you have an outside person interested in your progress who knows programming (such as a mother who is a software engineer with extensive Python experience), please make sure that you keep her at arms length. We can program the tasks ourselves and do not need her help in solving them. The tasks are intended to challenge you, make you think, and learn from your mistakes. If such a person insists on helping you, I recommend using Python to build a Django website or towards the end of the course, a gaming app in Python together. You will not learn problem solving or programming by watching others, you will need to make your own mistakes and recover from them in order to become efficient.

The laboratories are opportunities for you to solve larger tasks in groups. The normal student group will not be able to solve all of the problems; the additional problems are there to allow good students to become even better. Your lab score will only depend on solving the core problem successfully. The laboratory exercises are best done in pairs.

Peer based programming was once a paradigm to increase software productivity because practitioners realized that avoiding mistakes saves time over fixing them. A second pair of eyes and even more interaction with a peer allows programmers to discover flaws in their thinking or writing while code is being developed. While programming in pairs has turned out to be not cost efficient compared to other methods to improve programmer accuracy and is now only used in a few organizations, it can be a very efficient method to improve learning. If you want to try it, then designate one person as the **driver** and the other person as the **guide.** The driver types according to the instructions of the other, but can argue a strategy or an instruction. Every 10 minutes or with every task, the roles change. Please make sure that you do not get stuck in one rôle.

Grading

Grading is based on a number of different scores:

Daily quizzes:	
Individual quizzes	5%
Group quizzes	5%
Activities	10%
Four projects	20%
Laboratories	10%
Two Midterms (15% each)	30%
Final	20%

You need to achieve at least a score of 50% in **each** of the written exams in order to pass. Because individual learning times differ, there will be a make-up midterm whose score will replace the worst score of the two midterms. This make-up midterm will be given in the evening (outside of class time) and is voluntary. For each midterm and final, there will be a preparation sheet and a question and answer hour in the evening, outside of normal class time. Attendance for the Q&A hour is voluntary.

Grading Scale:

Grades will be assigned on a curve. However, students with 95% of the weighted average score are guaranteed to receive an A, those with 90% of the weighted average an A-, and students with 75% of the weighted average of scores are guaranteed to receive a C.

Course Policies:

There is no make-up for lost quizzes and activities. To accommodate student athletes and other students who miss for a legitimate reason, we will allow to make up four days' of timed grades (quizzes, activities, labs) using alternative means such as voluntary programming assignments. These will be made available to all, who can replace their lowest grades according to the instructions that come with each of these alternatives.

Because D2L cannot handle large sets of grades and large sets of students, daily and weekly grades will be reported only as summaries on D2L.

Course related communication is via the student's official email address. It is the responsibility of the student to check their emails.

Plagiarism in the age of stackoverflow and a plethora of Python blogs needs to be redefined. Students are supposed to learn how to use outside sources and normal use of these sources is permitted. The general rule is that all outside help going beyond a simple stackexchange answer or sample code from a python blog has to be documented in submitted work. If the outside source solves the problem or a substantial part of the problem, then the solution might not be used to fulfill a requirement. In this case, the student will be ask to resubmit a new solution or solve an alternative task. The decision whether this is the case lies with the instructor. However, if a student has documented outside work in a clear and efficient manner, then this use can never constitute a case of plagiarism.

It is now almost impossible to find exercises or projects for which a solution does not exist on the Internet. We recommend staying away from these solutions. Often, they are not very good or faulty. We are of course aware that it is possible to contract software development work for cheap via various Internet sites and that people have relatives in the industry. To be acceptable, your solution needs to be yours. We will ask you to explain any project solution and if you cannot do so, then we assume that the submission does not constitute a student's work and the student will not receive any credit for it.

The traditional definition of plagiarism applies to quizzes, midterms and finals and can be found in the student handbook. The policies outlined there apply.

The rules for administrative dis-enrollment will be applied to students who fail to attend class. If you do not submit a quiz, then you will be considered to have not attended the class.

Faulty and late projects can be resubmitted once a week after the deadline with a 25% discount in points.

We will monitor student progress and might have to change the rules of the course in order to adjust in the benefit of the students.

Rationale and Connection to the Creative and Technology and the Language, Cognition and Memory theme

A beginning programming class in computer science is not about learning the syntax of a language, but about learning how to use one's creativity in order to solve problems in an algorithmic manner. At the root of algorithmic design is creativity. One of the defining characteristics of a computer scientist is the desire to solve problems for people, problems that arise from human endeavors, whether it is the pursuit of beauty, subsistence, or intellectual achievements. Even in a beginning programming class, these almost universal aspirations enter, mainly through the example problems that are given to the students.

While we argue that this class is about creativity in problem solving, it is also an introduction to a technology that has, is, and will trans form the world.

The class is designed along the principles of “Learning in Teams,” where collaboration and communication are integral problem-solving skills. Students will develop their creative problem-solving and collaboration skills through activities — problems to be solved in pairs —, group quizzes, individual weekly small projects, larger sets of activities in the laboratories, and group large projects.

These learning activities will ask students to process large texts and corpora of textual and numerical data, to create interactive command-line applications, and to develop graphical interfaces using event-driven programming. For the latter, aesthetic criteria need to be developed and applied.

This class is using course material developed jointly with Xavier College (Autonomous) in Ahmedabad, Gujarat, India and the Xavier Institute of Engineering in Mumbai, Maharashtra, India.