Rationale for the department

The proposed course, currently referred to as CPSC203, will serve as a second course for novice programmers. Designed for non-CS majors, it emphasizes practicality and productivity, without compromising the essentials of computer science—discussion of algorithmic efficiency and software engineering are woven into the curricular narrative throughout the course.

Student background:

In the long term, we anticipate that students enrolling in CPSC203 will come from a variety of introductory experiences, including even informal or secondary school exposure to the fundamental constructs of programming. At its genesis, the course relies on the level of programming fluency achieved in CPSC103. The learning objectives of PHYS210: Introductory Computational Physics (http://laplace.physics.ubc.ca/210/Topics.html), suggest that it provides a reasonable foundation for CPSC203, as well, though some remediation in software design, and in Python syntax, may be necessary. In addition, COMM337: Business Programming and Analytics (http://blogs.ubc.ca/genemoolee/files/2018/01/COMM-337-Course-Outline-2018.01.02.pdf) would provide students with adequate preparation for the new course, though I am concerned that the pace of that course and magnitude of its syllabus would leave students with subpar mastery of its learning objectives. Finally, MATH210: Mathematical Computing (https://github.com/ubc-math210/2018September) is a fantastic pre-requisite. It gives a robust treatment of introductory topics, and it explores the domain of elementary numerical computation. We will continue to evaluate alternative introductory experiences on campus as we become aware of them.

While CPSC203 is designed to be a terminal course, there are a variety of follow-up options for those who wish to deepen their study of computing. At the end of the course, students will be prepared for applied data science courses like the new CPSC330, and they could also pivot to future non-major offerings such as courses based on the IoT, web programming, or mobile app development. **CPSC203 is not intended to be a pre-requisite for any major-stream CPSC course.**

Pedagogy and sustainability:

Course classroom instruction is designed as a mixed model of guided and exploratory lab work (we will be studying the efficacy of this approach), and as such we expect to facilitate laptop use in class, via some means, yet to be determined. We'll use Python 3, enhanced by popular libraries such as Pandas, Matplotlib, NLTK, and PILLOW, and including more advanced features of the language such as list comprehensions, basic OOP, unit testing, and type hints. Student deliverables include 4 substantial and increasingly independent programming projects, and approximately 60 fluency exercises, one per day.

The primary vision for sustainability revolves around a centralized course repository that would contain, in separate sub-repos, 1) the unchanging curricular materials, 2) the semester-by-semester creative curricular components, 3) a guidebook that should be used as a reference for both pedagogy and infrastructure, and 4) the general set of bookmarks and references that drive the content.

Some course curricular content would be slow to change, and updates would likely go through me, while others would vary every semester according to the preferences of the instructor. The breakdown of this by course element:

- Exams: There will be a template attached to learning objectives, but the instructor would build out content.
- Classroom exercises: unchanging (minimal course credit)
- POTD: unchanging (minimal course credit)
- Projects: I would like for these to be done collaboratively, between me and the instructor, at least until the template for them clarifies.
- Lab exercises: unchanging (minimal course credit)

A new instructor for the course would have creative responsibility for exams and projects. In addition, they would have the opportunity to create new classroom modules based on a strong narrative, and in discussion with me. (I envision these as a sort of pull-request to the course repo.)

It is naive to imagine that the background references and computing infrastructure will be static. Incremental changes to each will occur inside a "teachers guide" which I envision as a wiki space.

I will not know, until I build out the course content, whether or not additional student resources should be created. I am open to the idea of recording technical components of the course instructional narrative, for student reference (and possibly for pre-class work), but I cannot yet articulate which specific topics will be appropriate for that medium.

Conclusion:

Situating the new course is easy, given the summary presented in the Epilogue of HtDP—Onward, Developers and Computer Scientists (https://htdp.org/2018-01-06/Book/part_epilogue.html), which I believe is a fitting epilog to *any* robust introductory experience.

> "Right now, you might be wondering what to study next. The answer is both more programming and more computing.
>
> As a student of program design, your next task is to learn how the design process applies in the setting of a full-fledged programming language. Some of these languages are like the teaching languages, and the transition will be easy. Others require a different mind-set because they offer means for spelling out data definitions (*classes* and *objects*) and for formulating signatures so that they are cross-checked before the program is run (*types*).
>
> In addition, you will also have to learn how to scale the design process to the use and production of so-called frameworks ("stacks") and components. Roughly speaking, frameworks abstract pieces of functionality—for example, graphical user interfaces, database connections, and web connectivity—that are common to many software systems. You need to learn to instantiate these abstractions, and your programs will compose these instances to create coherent systems. Similarly, learning to create new system components is also inherently a part of scaling up your skills.
>
> As a student of computing, you will also have to expand your understanding of the computational process. This book has focused on the laws that describe the process itself. In order to function as a real software engineer, you need to learn what the process costs, at both a theoretical level and a practical one. Studying the concept of big-O in some more depth is a first, small step in this direction; learning to measure and analyze a program's performance is the real goal because you will need this skill as a developer on a regular basis. Above and beyond these basic ideas, you will also need knowledge about hardware, networking, layering of software, and specialized algorithms in various disciplines."

Effects of the course on enrollments of core major CPSC courses will be minimal. It is intended as a course for only non-majors. I anticipate that students who take the course will be interested in potential

future offerings of new computing courses like Applied Machine Learning (DSCI330 or CPSC330), Applied Database Design, Applied Scientific Computing, or Applied Information Visualization, but we do not expect that they will be prepared for the current CPSC versions of those courses. (We should expect students to request those CPSC courses, however, and fashion a consistent response.)

Course Design Resources (bookmarks):

1. http://interactivepython.org/runestone/static/thinkcspy/index.html
2. https://www.python-course.eu/advanced_topics.php
3. https://towardsdatascience.com/billboard-hot-100-analytics-using-data-to-understand-the-shift-in-popular-music-in-the-last-60-ac3919d39b49
4. https://github.com/guoguo12/billboard-charts
5. https://github.com/jdan/Melopy
6. http://www.martinbroadhurst.com/bin-packing.html
7. https://www.americanscientist.org/article/first-links-in-the-markov-chain
8. http://willdrevo.com/fingerprinting-and-audio-recognition-with-python/#
9. https://ieeexplore.ieee.org/document/7867412 (traffic simulation using CA)
10. http://www.aclweb.org/anthology/P10-1015 (social networks in novels)
11. https://www.acm.org/binaries/content/assets/education/cs2013_web_final.pdf

THE UNIVERSITY OF BRITISH COLUMBIA

Course Outline

Category: (1)

Faculty: Science **Department:** Computer Science **Faculty Approval Date:**	**Date:** 11 December 2018 **Contact Person:** Patrice Belleville **Phone: 2-9870** **Email:** patrice@cs.ubc.ca
Effective Date for Change: 19W **Proposed Calendar Entry:** CPSC 203 (3) Practical Programming (or Applied Data Structures and Algorithms) Students tackle increasingly complex algorithmic problems, using a modern programming language and a variety of approaches. Emphasis on problem decomposition and abstraction guide explorations of topics from applied algorithms, for example Voronoi Diagrams, Markov Chains, Bin Packing, and Graph Search. [3-0-1] Prerequisite: One of CPSC103, CPSC 110, MATH 210, PHYS210, COMM337.	**URL:** **Present Calendar Entry:** None **Type of Action:** Create new course. **Rationale for Proposed Change:** The proposed course, currently referred to as CPSC203, will serve as a second course for novice programmers. Designed for non-CS majors, it emphasizes practicality and productivity, without compromising the essentials of computer science—discussion of algorithmic efficiency and software engineering are woven into the curricular narrative throughout the course. In short, it answers the "now what?" question students ask after they complete an introductory course. **Not available for Cr/D/F grading.** (Check the box if the course is NOT eligible for Cr/D/F grading. Note: Not applicable to graduate-level courses.) **Rationale for not being available for Cr/D/F):** **Pass/Fail or Honours/Pass/Fail grading** (Check one of the above boxes if the course will be graded on a P/F or H/P/F basis. Default grading is percentage.)

1. Course Information:

Calendar entry: CPSC203 (3) Practical Programming (or Applied Data Structures and Algorithms)

Students tackle increasingly complex algorithmic problems, using a modern programming language and a variety of approaches. Emphasis on problem decomposition and abstraction guide explorations of topics from applied algorithms, for example Voronoi Diagrams, Markov Chains, Bin Packing, and Graph Search.

Prerequisite: One of CPSC103, CPSC 110, MATH 210, PHYS210, COMM337.

A. Expanded Course Description

At the completion of this course, students will view computation as a tool for solving complex quantitative, data-centric, or analytical problems. They will be fluent in Python. The problems addressed in the course arise from a variety of contexts, rely on diverse data sources (including APIs, text, .csv files, etc.) and produce appropriate output (including histograms and line graphs, images, music, data structure viz, etc.). Broad themes in the course include: 1) solving multi-stage problems using data design, abstraction, and decomposition, 2) algorithm efficiency, 3) data structures, 4) using computation to express ideas—bridging the gap between ideas and implementation.

The course consists of a sequence of 7 activities we call "explorations." Each exploration is a rich problem designed to expose course learning objectives in the context of an application. Classroom exercises and demonstrations are used to model relevant problem-solving strategies. Some explorations are derived from interesting data sources, others are puzzle-based, still others arise from classic problems in science or computing. Most have a breadth component whose purpose is to tie the exploration to applications outside the classroom.

The table below lists the explorations, broken into broad conceptual units: Programming Fluency, Applying Classic Algorithms, Algorithm Design, and Data Structures. Students will complete a project for each of those units, beginning with a constrained and guided analysis of hit streaks within the Billboard Hot 100 song rating system, and concluding with visualization of a relational or hierarchical structure created on top of a data set of their choice.

Exploration	Time	Short Description
PROGRAMMING FLUENCY		Sample Student Project: Billboard Hot 100 Song Streaks
1 Billboard Hot 100: Intro and Review https://www.billboard.com/charts/hot-100	1wk	We will use the billboard.py API to access and analyze various songs and artists over time. Questions may initially involve only a single top-100 list, but will expand to include multiple types of lists, over many weeks/years.

2 Handcraft: Iteration Games	1wk	Students will solve and design pattern puzzles, inspired by traditional handcrafts, and world flags.
APPLYING CLASSIC ALGORITHMS		Sample Student Project: Modeling Traffic w/ Cellular Automata
3 The Overstory: Discrete Voronoi Diagrams	1.5wk	Voronoi diagrams are commonly used to simulate the growth of forest overstory. We implement a simple nearest center algorithm, discuss simulation more generally, and explore additional applications of Voronoi diagrams.
4 Random Song Generator: Markov Chains	1.5wk	Ada Lovelace foretold the use of a computer for automatic music generation. We explore a model for music generation using Markov Chains. We also discuss additional applications of Markov Chains, including protein folding and page rank.
ALGORITHM DESIGN		Sample Student Project: Campsite scheduling
5 Knapsack: Hard problems	2wk	We explore the idea of computationally infeasible problems via a very light intro to NP-Completeness. Students devise their own approximation algorithms, and apply them to our test cases.
DATA STRUCTURES		Sample Student Project: Tournament or Social Network Viz
6 Volleyball tournaments: Trees	2wk	Given a set of volleyball game results, infer a tournament schedule. In the context of this exploration, students grapple with the complexity of recursive algorithms. We also challenge students to create a novel visualization of a tournament.
7 Harry Potter's Social Network: Graphs	2wk	We use co-occurrence of characters from text to infer Harry Potter's social network. The same algorithm is then applied to a variety of texts. Results are visualized via an interactive graph.

Principles from CPSC103, including data design, use of type hints, and testing, underlie coding practices in CPSC203, but we will not use custom libraries to support these practices. All libraries used in the course are either small open source projects, like Melopy (https://github.com/jdan/Melopy) or billboard.py (https://github.com/guoguo12/billboard-charts), or they are broadly adopted standard libraries, like matplotlib or nltk.

The course is intended for non-CS majors, and non-CS graduate students.

For the first offering, in 2019W1, we intend to start with 80-100 students. We anticipate that the course will scale up with CPSC103 enrollment, and that it will be appealing to students who come to university with a taste of programming from secondary school.

B. Course Objectives

Exploration 1 objectives: Students will:

1. install Python3 (anaconda) and open source IDE Atom on their personal laptops, if available.
2. use basic Unix commands, including cd, ls, pwd, mkdir, etc. to organize their work.
3. use source code control using Git.
4. use an API to retrieve data.
5. design data definitions relevant to a task.
6. speculate on their own questions to ask of a data set.
7. iterate over a list, filtering by a given attribute.
8. use nested iteration: for each list, for each week.
9. deploy Python's unittest framework on given test cases.
10. use Matplotlib to visualize multiple data streams.

Themes: review and setup, iterative assembly of data, using APIs, testing and debugging.

Exploration 2 objectives: Students will:

1. explore RGB color representation in the context of general data representation.
2. use PILLOW (a Python imaging library) to access image data.
3. iterate over the pixels in an image applying myriad photo filters of their own design.
4. define algorithmic complexity and analyze image modification algorithms.
5. create new images as composites and functions of old images.
6. create visual patterns using functional decomposition of image segments.
7. design a set of functions that could be used to define a collection of flags of world countries.

Themes: more complex iteration, problem decomposition, images as data.

Exploration 3 objectives: Students will:

1. be able to explain how the overstory of the forest can be simulated by Voronoi diagrams.
2. be introduced to the dictionary data structure.
3. solve many small programming problems related to dictionary syntax.
4. practice transforming data from, for example, lists of occurrences into a dictionary of frequencies.
5. design an algorithm for computing Voronoi diagrams given a set of points, using dictionaries.
6. analyze the complexity of an algorithm to compute Voronoi diagrams.
7. be introduced to Python objects in the context of points in the plane.
8. use random numbers to generate simulated data.

Themes: classic algorithms applied to scientific problems, basic data structures, objects, complexity.

Exploration 4 objectives: Students will:

1. be able to describe Ada Lovelace's prescient description of a computational device to compose music.
2. use Melopy, a simple music library for Python, to generate simple songs.
3. write code to count co-occurrences of notes in a song.
4. write code to simulate song composition using Markov Chains.
5. solve simple problems using discrete probabilities.
6. compute steady state probability distributions.

7. describe how Markov chains can be used for computer simulations.

Themes: simulation, more complex iteration and data assembly, basic discrete probability.

Exploration 5 objectives: Students will:

1. be able to describe computationally infeasible problems, especially the bin-packing problem.
2. design and implement approximation algorithms for bin-packing.
3. comfortably manipulate data into lists of dictionaries containing list elements, and other complex data definitions.
4. challenge their approximation algorithms by finding inputs for which the approximation is not optimal.
5. practice critical reading of problem statements, contrasting bin-packing with efficiently solved similar problems.
6. visualize bin-packing approximate solutions.

Themes: Problem statements, intractability, approximation algorithms, increasingly complex data manipulation.

Alternative Exploration 5 objectives (instructor choice): Students will:

1. be able to explain how music identification software works.
2. explore FFT and music spectrograms.
3. use peak amplitudes to fingerprint songs.
4. characterize music fingerprints using a hash table.
5. use a simple Python SQL database.
6. search a database of hashed music fingerprints to match a given query.

Themes: Sophisticated data manipulation, hashing, introductory SQL.

Exploration 6 objectives: Students will:

1. recursively define hierarchical structures (trees)
2. design and implement an algorithm to infer a tournament tree from a set of completed games.
3. implement the breadth first traversal of a tree.
4. search a tree for a given team's results (application of traversal).
5. design and implement a novel visualization of a tournament tree.

Themes: the tree data structure, working with Python's tree implementation, algorithm design.

Exploration 7 objectives: Students will:

1. explore nltk, Python's Natural Language ToolKit, through simple exercises.
2. use nltk to parse text from a novel into lists of words.
3. use nltk to extract character (or other named entity) information from the text.
4. tally co-occurrence of characters in small segments of the text.
5. interpret character co-occurrence as the edges of a graph.
6. explain graph implementation, generally and in Python.
7. visualize social network graphs of characters in a novel.

Themes: solving a problem with many steps, graphs and graph implementation, text as data.

Alternative Explorations 6 and 7 may be developed based on data from Pokémon. Specific learning objectives for those explorations depend on the API used. We have conducted discussions with Niantic,

the Pokémon Company, and anticipate that Niantic will be opening its API for educational purposes sometime in the near future.

The overriding purpose of the course is to give students the tools they need to formulate and answer questions that arise from their own interests and study domains. We plan to give them lots of joyful examples on which to build their knowledge base.

C. Potential Instructors

Cinda Heeren, Kemi Ola, Mike Gelbart, Meghan Allen, Paul Carter.

2. Course Format:

Classroom:
3x50min activity-based classroom meetings per week. Example activity types include: 1) pair-programming solutions of problems related to the course projects, 2) worksheet-based creativity exercises designed to help them to understand data or articulate problems, and 3) interactive demos that introduce problem domains.

Problems of the Day (POTD):
Students will be given a daily homework assignment in the form of a programming task whose solution should require no more than 15min (five per week, 60 for the term). These exercises will increase fluency and provide practice in subtasks necessary for project success. They will be based on the assigned reading, and they will serve as pre-class activities. We expect to use a web-based programming interface such as that found at http://repl.it/ for this portion of the course. These problems will emphasize coding conventions and algorithms using simple data (mostly strings and numbers).

Labs:
Lab exercises will serve as a middle ground between the "small-thinking" necessary for solving the POTD, and the "large-thinking" necessary for solving the course projects. Skills related specifically to Python programming, such as list comprehensions, class definitions, and type hints will be introduced in lab exercises. The syntax of course libraries will be addressed in lab, freeing classroom time for more conceptual work. Complete lab exercises will frequently serve as templates for solving components of the larger project. For example, Lab 2 will instruct students in the use of Matplotlib to create line graphs for multiple data streams, and they will then use that skill to display the results of their computation in Project 1.

Timeline:
A typical week in the course requires the following engagement by students:

Activity	Duration
Classroom meetings	3x50min
Lab	1x50min
POTD	5x15min
Pre-Class reading	3x20min
Project development	4hr

Total	9.5hr

Marking protocols:
Lab exercises will be self-assessed in conversation with the lab TA. Lab solutions will be available for independent evaluation, but no TA marking will take place.

Problems of the day will be automatically marked by the online delivery system. Students may resubmit their work repeatedly to achieve credit.

Each project will be comprised of a technical component (2/3credit) and a creative component (1/3credit). The technical component will be marked via automated testing against provided test cases. The creative component will be TA and peer evaluated during lab meetings using a rubric provided when the assignment is released.

Computing Infrastructure:
Students will deploy Python3 (anaconda) on their own laptops, if possible. Solutions for students without laptops are being explored. GitHub will be used as the platform for all code distribution and evaluation. Best practices for remote repository use will be taught early in the course and will be expected throughout.

3. Course Schedule:

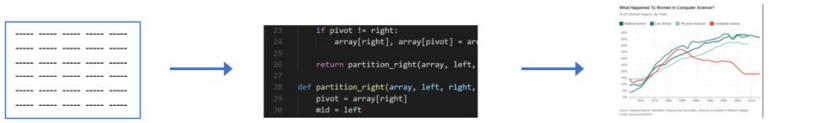
See table above for classroom and project schedule. There will be a midterm after unit 4, at the midpoint of the term. A final exam will be used to assess design processes and programming fluency.

4. Learning Outcomes:

Many, many, programming tasks involve recognizing and articulating the data transformations that must take place in order to solve a problem. The goal of this course is to provide students with a broad set of skills they will deploy when addressing computation problems of their own, and from their own domain.

In CPSC103, and in most other introductory courses, students learn principled and robust techniques for solving simple linear problems:

Typical Introductory Data Flow:



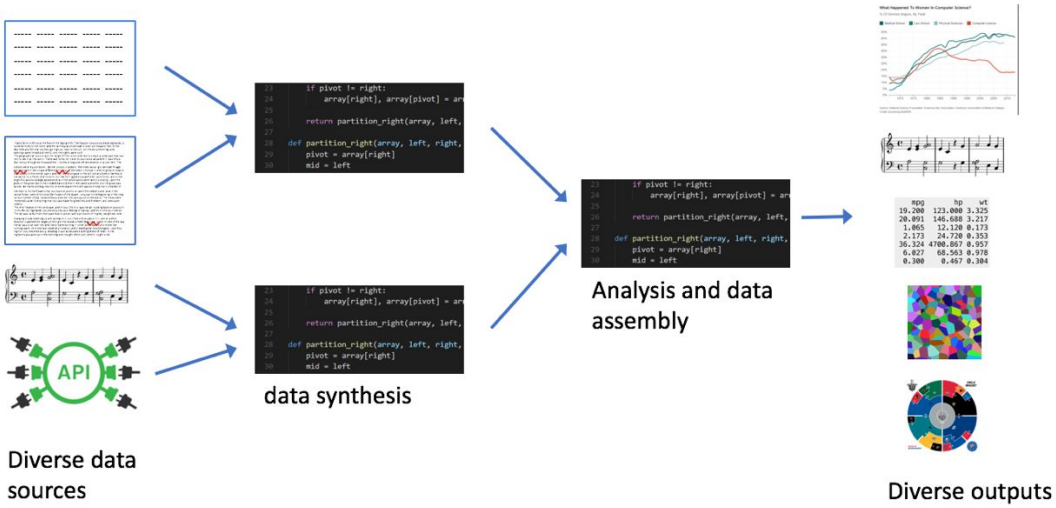
.csv file

Python problem solution using simple data types and elementary list iteration.

Matplotlib bar or line graph or other summative output illustrating results of computation.

The proposed course, CPSC203, takes every element of that problem flow and introduces more complexity and more variety:

CPSC103++ Data Flow:



Diverse data sources

data synthesis

Analysis and data assembly

Diverse outputs

5. Assessment Criteria and Grading:

Students will be graded on a numerical basis.

Evaluation

Grading scheme (percentage):

Programming Projects	4 x 7.5% = 30%
Problems of the Day	60 x 0.25% = 15%
Lab Exercises	10 x 1% = 10%
In-Class Exercises	5%
Midterm	15%

Final Exam	25%

6. Required and Recommended Material:

Daily exercises and reference to particular constructs and syntax will come from:
http://interactivepython.org/runestone/static/thinkcspy/index.html

All other course materials will be instructor-authored.

7. Articulation:

Once approved, the course description will be forwarded to Computer Science departments at other BC institutions and they will be encouraged to submit articulation requests for any courses they offer that they feel are similar to CPSC203.

8. Budget Impact:

This course will require one instructor and the same number of TAs that other undergraduate CPSC courses are assigned. The department will assume any costs that might result if we see an overall growth in the number of second-year students that we teach.

9. Library Impact:

None

10. Consultation With Other Academic Units:

We will consult with ARTS, COMM, CPEN, MATH, PHYS and STAT.

***Statement of Academic Integrity*:**

NOTE: All instructors have been asked to include the following type of statement of Academic Integrity in their course outlines distributed to students.

The academic enterprise is founded on honesty, civility, and integrity. As members of this enterprise, all students are expected to know, understand, and follow the codes of conduct regarding academic integrity. At the most basic level, this means submitting only original work done by you and acknowledging all sources of information or ideas and attributing them to others as required. This also means you should not cheat, copy, or mislead others about what is your work. Violations of academic integrity (i.e., misconduct) lead to the breakdown of the academic enterprise, and therefore serious consequences arise and harsh sanctions are imposed. For example, incidences of plagiarism or cheating may result in a mark of zero on the assignment or exam and more serious consequences may apply if the matter is referred to the President's Advisory Committee on Student Discipline. Careful records are kept in order to monitor and prevent recurrences.