# CS 101 - Learning to Code at BYU

## **Description**

Here at BYU, CS 142 (Introduction to Computer Programming) is a course that tends to show students who are interested in coding just how difficult coding can be. Students who initially show great interest in the subject may flounder in the course if they do not get significant help from the instructor and the TAs, especially if the students had little to no coding experience prior to the course. Their interest and their motivation slowly wither away as they come to learn that coding is too hard for them and should be left to computer scientists. I want BYU to have an introductory programming course that does not focus as much on rigor as it focuses on encouraging interest and exploration.

## **Goal**

I want to design a course that is for BYU students with little to no coding experience who want to learn how to code. I want the course to not be too overwhelming for beginners, and I want it to encourage exploration and help students to be excited about what coding can do for them. I don’t want students who initially show interest in coding to feel like coding is too hard or to get overwhelmed and burn out. I want it to be a positive experience that shows students that they can code and that coding can help them do things that they are interested in. I want to provide students with a wide range of assignments and multiple paths of completion so that students can write code that interests them. This goal satisfies the desire for change by providing students a way to learn coding that focuses on keeping students motivated by keeping it simple and allowing the students to explore what they find interesting about coding.

## Success Criteria

**Project success (what clients and stakeholders care about).** The major stakeholders are students, potential instructors (including myself), and the CS department.

Students who enroll in the course will care about doing well in the course (low withdrawal rates and high passing rates), feeling confident in their abilities, being excited/motivated about the course material and the assignments, and learning skills that they feel are relevant to them.

Potential instructors care about positive student ratings. Instructors will want to see students improve and help students to be excited about learning to code.

The CS department cares about its national ranking, and current top-ranked CS programs offer introductory courses for non-majors, so my hope is to help the CS department to care more about offering such a course.

**Product success (what students are learning).** The product will be successful if students learn how to write code that they feel is relevant to them. Students should attain the learning outcomes  (see below).

## Content Layer Model

The content model consists of both expert performance and systems of cause/effect rules.

- Expert performance
  - Students will learn from experts how to
    - Store primitive data types as variables
    - Manipulate variables

        - Use control structures (if/then/else, switch statement, etc.)
        - Use iteration (for, while, do-while)
        - Use arrays/lists and other containers (tuples, sets, dictionaries, etc.)
        - Design and call functions
        - Define classes and create objects
- System/environment rules
    - Syntax rules
        - Students should understand the syntactic rules of the programming language. What are the keywords (or reserved words) in the language? How are variables and functions named? How are functions called? How are commands separated? How does one find and fix syntax errors in code?
    - Semantic rules
        - Students should understand the semantic rules of the programming language. Some bugs occur when the code is syntactically correct, but a command is not doing what the programmer thinks it is doing. What is the scope of variables and functions? What is the difference between passing by reference and passing by value, and when should these techniques be used? How does one find and fix semantic errors in code?
    - Logic rules

        - Students should understand basic logical structure. Some bugs occur when the code is both syntactically and semantically correct, but the programmer has made a logic error in code, such as
            - Failing to account for every possible use case (especially corner cases)
            - Improper sequencing of commands
            - Failed to account for race conditions
    - UX rules
        - Students should understand how to write code to provide a positive user experience. For example, if a program requires a number as input and the user inputs a string containing non-digit characters, the program should be written to handle the error gracefully rather than crashing.

## Constraints

- CS department approval
    - CS department wants to focus on curriculum for CS majors to improve the CS program's national ranking, so they have voted against having a CS class for non-majors.
    - CS department is not interested in recruiting more CS majors; they already have more than they can handle. They like that CS 142 weeds them out.

    - Not all faculty agree. Some would like to see an introductory coding course for non-majors.
- If approved by the CS department, the course would likely not be able to be held in a computer lab. Most CS courses aren't held in computer labs due to class size and lack of resources. Students would have access to the CS computer labs outside of class, but the ideal scenario would be to teach the class in a computer lab so that the instructor can dedicate some class time to helping students with their code.
    - However, I did have one CS class taught in a computer lab on campus (in the then-new Life Sciences Building), so this isn't necessarily going to be a constraint.
    - Another way around it would be to design a BYOD course. Students without laptops could rent laptops from the university.

## Learning, Instructional, and Design Theories

- Computer supported collaborative learning (CSCL)
    - The LMS for the course will allow students to communicate together and share code for assignments that are past due. In my own experience, I have learned a lot from sites that have coding challenges that allow you to see others' code for each challenge after you solve it yourself. It's a great way to learn the features of a language and efficient algorithms.
- Gagne's nine events

    - The emphasis of class time will be providing learner guidance and eliciting performance, but all nine events will be used in class. There will have to be some lecture, and each topic builds off the last, so there will be a need to recall prior learning before launching into new topics.
- Tom Sticht's functional context
    - Instruction should be made as meaningful as possible to the learner in terms of the learner's prior knowledge.
    - The materials and equipment used in class will continue to be helpful after the students finish the course.
        - This is one of the reasons I decided on Python as a programming language over something like Scratch or Blocks. You can write/run Python code in more environments.
    - Valid assessment of learning requires context/content specific measurement.
        - Assessments will focus on students' abilities to read, write, and debug code.
- Problem based learning (PBL)
    - At first, the assignments will be simple and will basically look for a single solution. As the course moves on, however, problems will become more open-ended and students will have to architect their own solutions to real-world problems.
- Merrill's first principles
    - Learning is promoted when learners are engaged in solving real-world problems.

        - Problem based learning
    - Learning is promoted when existing knowledge is activated as a foundation for new knowledge.
        - Each new topic is dependent on the last
        - Use of metaphors and examples from the real world to explain complex programming paradigms
    - Learning is promoted when new knowledge is demonstrated to the learner.
        - Live coding will be part of each lecture
    - Learning is promoted when new knowledge is applied by the learner.
        - Vast majority of assignments will be coding assignments
    - Learning is promoted when new knowledge is integrated into the learner's world.
        - Students can choose from wide variety of possible assignments so that they can work on things that they find interesting or relevant to their everyday lives.
- Gibbons's design layers
    - Think of online materials and in-class lecture in terms of strategy, content, messages, controls, representations, media-logic, and data management
- Backward course design
    - Design learning outcomes first, then assessments, then coding assignments, then reading assignments and lectures. This helps to focus everything on the learning outcomes.
- Bloom's Taxonomy

    - This course will focus on the “understand” and “apply” layers of Bloom’s taxonomy.
- Cognitive apprenticeship
    - Teaching computer science is sometimes difficult because it is hard for subject matter experts to empathize with beginners. Experts forget to explain basic implicit tasks. Cognitive apprenticeship is designed to bring those tasks into the open so that students can observe and practice with help from the teacher
    - Teaching methods include modeling, coaching, scaffolding, articulation, reflexion, and exploration

## Strategy

### Programming Language

I decided to use Python for this course. Python is more human-readable than languages like Java and C++. It's not graphics-based like Scratch and Blocks, and I feel like graphics-based programming languages are too simplistic for what I want this course to do. Python seems more age appropriate. JavaScript is a close second choice for me, but there is some pretty odd JavaScript syntax that I don't want to get into, and JavaScript doesn't support local file IO.

### Learning Outcomes and Objectives

- Outcomes
    - Self-efficacy. I want students to not be afraid of coding. I want them to realize that they can do it!
    - I want students to realize that coding relates to things that are important to them, even if they don't intend to pursue a degree or career in technology. I want them to be excited and motivated about learning to code.
    - I want students to be agents of their own learning, able to explore and learn on their own.
    - I want students to be able to work and communicate effectively with developers.
- Objectives
    - Coding skills
        - Print messages to stdout
        - Store primitive data types as variables
        - Manipulate variables
        - Get user input from keyboard
        - Design and call functions

        - Use control structures (if/then/else, switch statement, etc.)
        - Use iteration (for, while, do-while)
        - Use arrays/lists
        - File I/O
        - Use dictionaries (aka structs, JS objects, associative arrays)
    - Read others' code and determine what the code is doing
    - Find and fix bugs in code
        - Understand difference between syntax, semantic, and logic errors
    - Know basic coding terminology

**How My Strategies Lead to Learning Outcomes and Tie to Theory**

- Flipped classroom strategy
    - Online reading assignments and micro assessments
        - The flipped classroom strategy puts more responsibility on students to be agents of their own learning
        - Help introduce concepts to students and help them apply it in micro assessments
        - Micro assessments test student's knowledge of coding terminology, ability to read code, ability to find and fix bugs, and ability to apply new concepts in simple ways
            - Ties to behaviorist principles of learning by repetition
            - Ties to Merrill's first principles by activating existing knowledge, demonstrating knowledge, and having the learner apply the knowledge
            - Ties to learning objective of knowing the terminology and being able to communicate with developers

      - If a student does not perform well in micro assessments, the reading module dives deeper into the concepts that the student seems to be struggling with, then provides another assessment
        - Programmed instruction, behaviorism
  - Work on assignments in class
    - Near the end of the semester, when the problems get more complex and open-ended, students can collaborate together to come up with ideas for architecting solutions. It will be more PBL-based.
      - This ties to social constructivism as students collaborate together
      - This ties to Merrill's first principles by having students work on real-world problems
    - Ties to cognitive apprenticeship. Class time will be about modeling, coaching, scaffolding, articulation, reflexion, and exploration.
  - Short lecture (10-15 minutes) every class
    - Focus on subject matter that students seemed to struggle with (based on results of reading assessments and where students are in assignment tree)
- Why Python?
  - Python has human-readable syntax, which should help students be able to read it and know what's going on. My hope is that the learning curve is smaller because of the more intuitive syntax.
  - Python can be run on a wide variety of devices/environments and is a popular "real-world" programming language
- LMS strategies
  - Assignment tree

        - Students can be agents of own learning, choosing which assignments sound interesting to them. This should also help them feel more excited and motivated about the projects.
        - Students can go at a faster pace if they want to (agents of learning, self-efficacy)
    - Auto-graded assignments
        - Students get immediate feedback (behaviorism, Gagne's nine events)
    - Online reading assignments and micro assessments
        - See "flipped classroom strategy" above
    - Student discussion board
        - Allows for student collaboration (CSCL)
    - Leader board
- Test strategies
    - Take-home coding assignments
        - Tests will be like assignments, but more complex and collaboration with other students will not be allowed.
        - Students completing these bigger projects on their own will lead to self-efficacy
        - Assessments directly tie to learning objectives, because learning objectives are about coding skills

**Stakeholder Input**

- CS Department, Instructors
    - "Programming language doesn't really matter; it's just syntax. The concepts are the important thing to focus on."
        - I agree with this in some ways, but disagree in other ways. The concepts are important to focus on because they can transfer to other programming languages.

However, languages are more than just syntax; some languages have features/concepts that other languages don't have. Also, I think that human-readable syntax can go a long way in making code more approachable for beginners.

        - I made the learning objectives language agnostic so that if an instructor wanted to teach this course with another programming language, that instructor could do so without changing the design of the course.
    - "The university needs an introductory computer course of some kind. Students who take CS 142 without any coding experience struggle and flounder."
        - I removed some complex concepts (such as recursion and object-oriented programming) from the learning objectives so that the course could spend longer on each of the basic concepts.
- Students
    - "I don't have enough time between assignments to digest or play with what I've learned. It's just constant go, go go."
        - Again, I removed complex concepts from the learning objectives so that students could spend longer on each of the basic concepts.
        - Students can explore concepts in different contexts by completing a lot of small assignments rather than a single large assignment that requires deep usage of a single concept.
    - "There is a disconnect between the lectures, the assignments, and the tests. I wish the tests weren't so random and that the lectures focused more on helping us with assignments."

        - The flipped classroom paradigm will make class time about completing the assignments.
        - The tests will be take-home coding tests. Tests are basically just more complex assignments that students can't collaborate on.
    - "I was interested in coding, but the course just went too fast and I couldn't keep up."
        - The course will focus more on basics, and the assignment tree will allow for flexible pacing.
    - "I wish I could work on assignments on my own computer instead of going to the lab. I just don't have time to figure out how to install and set up all the necessary software."
        - The LMS will have a space for students to write, run, and test code, so they will be able to work on assignments on any device with a browser.
        - If students need to be able to code offline, there are Python IDEs that are a lot easier to install and configure than most Java or C++ IDEs.

**Implications on Other Design Layers**

- Partitioning of knowable content
    - See learning objectives
- Representation of instructional messages in sensory form
    - The reading is online and interactive
        - Online "reading" can be text on the screen, graphics, and video clips
    - Class time may include some power points, video clips, or live coding models
- Provision of controls for learner use
    - LMS provides controls for students to read material, complete reading assessments, write/run/test/submit coding assignments, and communicate in discussion boards
    - Students can ask questions in class

- Arrangement of execution means and logic
    - The flipped classroom and online assignment tree allow students to go at different speeds and to work on different assignments. Decisions of what to focus class lecture on will have to be made by reviewing LMS data (reading assessment performance and assignments turned in)
- Gathering of data and its use in tailoring instruction to the individual
    - Online reading and micro assessments store student responses and adapt based on student's needs
    - Instructors and TAs will work with students in class and observe where students are and in what areas they are struggling. They will also have administrative access to LMS to see how well students are progressing down the assignment tree and how well they perform on reading assessments.

**Testing/Prototyping Assumptions**

- Python is more intuitive and easier for beginners to grasp than a lower-level language like C/C++ or a complex object-oriented language like Java
    - This is the least worrisome of my assumptions. I have to pick a language, and Python isn't likely to be harder to work with than other languages. It may not be the absolute best language for this course, but it's probably good enough.
- Students will be more motivated and excited about assignments if they are able to choose from a set of possible assignments
    - Literature - Educational Psychology, 3rd ed - Allowing students to choose specific tasks or assignments for themselves encourages mastery goals and increases motivation (see online LIDT book, chapter on Motivation Theories on Learning)

    - I feel like there could be some unintended consequences from this decision. Maybe students will feel overwhelmed by multiple options and will have a hard time choosing which assignments to do. Maybe students will get started on one, get stuck and pick a different one, and end up with a bunch of half-completed assignments because they were hoping to find something better/easier.
- Students will take reading assignments seriously even though the reading assessments are low stakes
    - This is the assumption I'm most concerned about. In a flipped classroom setting, the instructor is expecting the students to get a lot out of the reading since it won't all be covered in class. Also, the reading assessments are critical in helping students to meet all of the learning objectives.
    - I could raise the stakes by randomly asking students to share what they learned in the reading or by doing spontaneous in-class quizzes, but that may increase stress and might use up more class time than I would like.

**Assessment**

- Reading assessments
    - Cumulative
        - Students will have to recall learning multiple times throughout semester, helping them to remember it and understand how concepts can apply in different contexts
    - Low stakes, formative
        - Not too stressful, students hopefully won't get too burned out or overwhelmed even if they do poorly on some reading assessments
        - Help students realize they can do this

    - Online instruction adapts based on student performance
        - Online instruction reinforces areas in which the student may be struggling
    - Test student’s knowledge of coding terminology, ability to read code, ability to find and fix bugs, and ability to apply new concepts in simple ways
        - Some multiple choice/matching
            - Used for vocabulary, classifying bugs, determining what code is doing
        - Some writing/modifying code to get it to work a certain way
            - Used to apply new concepts, find and fix bugs
    - Immediate feedback
        - Students know exactly how well they perform and how they measure up to expectations
- Midterm & Final
    - Summative
    - Take-home coding challenges
        - Learning outcomes are primarily about ability to write code, so making the students write code for summative assessments makes the most sense
        - Challenges will require use of multiple concepts
            - The final will require use of every coding concept defined in learning outcomes
        - Open book, open notes, open Internet as long as they are not copying code
            - Just like coding in the real world
    - No collaboration with classmates
        - Promote self-efficacy by making students do these on their own

# Syllabus

## **Course Information**

### **Description**

CS 101 focuses on the basics of computer programming. By the end of this course, you will be equipped to write code in Python to help you solve real-world problems that are relevant to you. All course activities are designed to provide practical experience in writing code to develop solutions to problems. No coding background is required or even recommended for this course; it is specifically designed as a course for beginners. The course is designed for students from a wide variety of majors who are interested in learning how to code. You will learn how to code using the Python programming language to

- Handle input and output
- Manipulate variables
- Use logic control structures
- Group data into lists
- Iterate over data
- Design and call functions
- Group data into dictionaries

### **Teaching Philosophy**

In this course, learning is placed in your hands. You are given a fair amount of agency—you can choose which coding assignments you want to work on and when to work on them. You are expected to set your own pace according to your own needs as you explore the material. You cannot be successful in this course if you procrastinate, so be proactive.

Reading assignments should always be completed before class time of the day listed on the schedule (see below). The instructors will **not** cover all of the material from the reading in class, so you will miss important content if you fail to do the reading. The most critical content from the reading will be briefly covered in class, but the majority of class time will be spent working on coding assignments. You will likely struggle with coding assignments and fall behind if you fail to do the reading on your own before class.

**Learning Environment**

Class will be held in a computer lab. You are welcome to use the lab computers or to bring your own. All reading assignments, coding assignments, and exams for this course are available on the course's online portal. You can authenticate to the online portal using your BYU Net ID and password. In addition to providing a place for working on and submitting assignments, the online portal can be used for discussion and collaboration. Students are welcome to use the discussion board to discuss readings, coding assignments (not exams), and programming concepts, but should never directly share code.

**Reading Assignments and Assessments**

Reading assignments are available through the "Reading" tab on the online portal. Each reading assignment has a specific due date, and must be completed before class on the due date to receive credit. The reading assignments are designed to teach you as you go. Throughout the reading, there are small coding simulations that help you to apply what you have learned. These small simulations are not graded and are not required in any way, they are meant only to help you with your learning. They can give you tailored feedback to help you improve.

At the very end of each reading assignment is a reading assessment. These assessments are graded. Assessments may include coding challenges and/or multiple choice questions. Each assessment is worth 10 points: 5 for completion and 5 for getting everything right the first time through. If you understand and complete the small coding simulations throughout the reading, you will be much more likely to get everything right. If you get an assessment question wrong, you will lose a point (up to five points) and be referred to the portion of the reading that will help you to learn the correct answer. You will then answer the question again, and you will lose another point if you get it wrong again. This pattern continues until you answer the question correctly. The reading assessment is complete once you have answered every question correctly.

**Coding Assignments**

Coding assignments are available through the “Coding” tab on the online portal. You are given some freedom to choose which coding assignments to do and when. To get full points, you must complete at least 30 of the possible coding assignments. Assignments are displayed in three different states: green assignments have been completed, orange assignments are available, and gray assignments are not yet available. To “unlock” a gray assignment to make it available, you must first complete the prerequisite assignments. To view an assignment tree showing all possible assignments flows, you can select the “Tree View” instead of the default “List View” at the top of the screen.

When you click on an assignment from the tree or list view, it will open up in the complete assignment view. The assignment view contains the full description of the assignment and a Python development area. The development area can be used to write, test, and submit code for the assignment. You may write and test your code in a different environment if you would prefer to (e.g., you may want to install a Python development environment on your local machine to work on assignments offline), but all code submissions must be done through the online portal. When code is submitted, it is automatically checked by the portal’s autograder, which tests your code with a variety of inputs and analyzes the output to see if it is correct. If your code fails any of these tests, the portal will show you the input used on one test that failed, and you will be allowed to fix your code and submit again. You may submit however many times you need to get your code right.

Each coding assignment is worth 15 points. 5 points are awarded by the online portal's autograder. If you submit a working solution, you will be awarded the full 5 points. The other 10 points come from the instructor's code review. The instructor will grade your code based on how well you used the material from the readings to complete the assignment. If the instructor detects any copied code (copied from classmates or online sources) or cheating of any kind (such as hard-coded input/output pairings intended to fool the autograder), you will receive a zero on the assignment and be pulled aside to talk with the instructor. If you receive fewer than 15 points on a coding assignment, you can make up the points by doing extra coding assignments (more than the expected 30).

Once you successfully complete an assignment, you will be able to view the solutions of the instructor and other students. You are encouraged to look over these other solutions to learn from your peers. If a section of code from these solutions can be used in future assignments, you are welcome to copy and paste as long as you (1) give credit to the original author and (2) provide a brief explanation of how and why the code works.

If for some reason you cannot complete an assignment and you wish to go on to other assignments, you may mark an assignment as complete and take a zero. You will then be given access to the solutions for that assignment just as if you had successfully submitted it. This can be a good strategic move in some cases, but be careful to not overuse it. The best way to learn how to code is to keep trying until you figure out how to get it to work on your own.

**Exams**

There are two exams in this course, a midterm and a final. Both will be online take-home problems that can be worked on and submitted through the online portal. Like other coding assignments, they will be open book, open notes, and open internet. However, you should not discuss the exam with anyone but the instructor, either online or in person. Exam solutions should reflect original individual work, and each submitted solution will be analyzed for originality. Anyone caught asking for or providing help in any online or offline discussions will receive a zero on the exam.

Unlike other coding assignments, each exam will only be available for a two-week window and must be started and completed during that time. When you submit an exam solution to the autograder, it will immediately give you a complete grade (out of 100 for the midterm, 200 for the final) based on how many tests your code solution passed. You can continue submitting more solutions up until the final due date. At that point, your **last submission** will be the only one recorded on your grade, even if it was not your best submission.

**Grading**

| | | |
|---|---|---|
| Reading | 250 points (25 assignments, 10 points each) | 25% |
| Exams | 300 points (100 midterm, 200 final) | 30% |
| Coding Assignments | 450 points (30 assignments, 15 points each) | 45% |
| **Total** | **1000 points** | **100%** |

| **Grade** | **Minimum Percent** |
|---|---|
| A | 93% |
| A- | 90% |
| B+ | 87% |
| B | 83% |
| B- | 80% |
| C+ | 77% |
| C | 73% |
| C- | 70% |
| D+ | 67% |
| D | 63% |

| | |
|---|---|
| D- | 60% |
| E | 0% |

**Other Resources**

You are welcome to use any resources you can find to learn Python. Here are a few that are tried and true. If you find another resource that you like, please mention it on the online discussion board and it can be added to the list.

- http://www.learnpython.org/
- https://www.codecademy.com/learn/python
- https://developers.google.com/edu/python/introduction
- https://www.python.org/about/gettingstarted/

**Schedule**

| Class Number | Reading & Discussion | Exams |
|---|---|---|
| 1 | Syllabus | |
| 2 | Variables, data types, and keyboard input | |
| 3 | Integers, floats, and mathematical formulae | |
| 4 | Booleans and truth functional logic | |
| 5 | Strings - basics | |
| 6 | Strings - formatting and functions | |
| 7 | Function design | |
| 8 | Function design | |
| 9 | Control structures - if/elif/else | |
| 10 | Control structures - nested if/elif/else | |
| 11 | Finding and fixing syntax, semantic, and logic errors | |

| | | |
|---|---|---|
| 12 | Iteration - while loops | Midterm Open |
| 13 | Iteration - while loops | Midterm |
| 14 | Iteration - for loops | Midterm |
| 15 | Iteration - nested loops | Midterm |
| 16 | NO READING | Midterm Closed |
| 17 | Lists - basics | |
| 18 | Lists - iteration | |
| 19 | Lists - functions | |
| 20 | Lists - nested lists | |
| 21 | File I/O | |
| 22 | File I/O | |
| 23 | Dictionaries - basics | |
| 24 | Dictionaries - functions | |
| 25 | Dictionaries - lists of dictionaries | |
| 26 | Dictionaries - nested lists and dictionaries | |
| 27 | NO READING | Final Open |
| 28 | NO READING | Final |
| Finals Week | Final due on last day of finals | Final Closed |

## Reading Assignments and Assessments

### List View

From the list view, users can select any reading assignment. Users are not forced to do the readings in order, and all assignments are available throughout the semester. Assignments are green if they have been completed, otherwise they are orange. The list view shows the name of the reading assignment, the due date, the user's completion date, and the user's score.

| # | Title | Due | Completed | Score |
|---|---|---|---|---|
| 1 | Syllabus | mm/dd | mm/dd | 10 |
| 2 | Variables, data types, and keyboard input | mm/dd | mm/dd | 10 |
| 3 | Integers, floats, and mathematical formulae | mm/dd | mm/dd | 9 |
| 4 | Booleans and truth functional logic | mm/dd | mm/dd | 10 |
| 5 | Strings - basics | mm/dd | mm/dd | 7 |
| 6 | Strings - formatting and functions | mm/dd | | |
| 7 | Function design | mm/dd | | |
| 8 | Function design | mm/dd | | |
| 9 | Control structures - if/elif/else | mm/dd | mm/dd | 10 |
| 10 | Control structures - nested if/elif/else | mm/dd | mm/dd | 9 |
| 11 | Finding and fixing syntax, semantic, and logic errors | mm/dd | | |
| 12 | Iteration - while loops | mm/dd | | |
| 13 | Iteration - while loops | mm/dd | | |
| 14 | Iteration - for loops | mm/dd | | |
| 15 | Iteration - nested loops | mm/dd | | |
| 16 | Lists - basics | mm/dd | | |

|  |  |  |  |  |
|---|---|---|---|---|
| 17 | Lists - iteration | mm/dd |  |  |
| 18 | Lists - functions | mm/dd |  |  |
| 19 | Lists - nested lists | mm/dd |  |  |
| 20 | File I/O | mm/dd |  |  |
| 21 | File I/O | mm/dd |  |  |
| 22 | Dictionaries - basics | mm/dd |  |  |
| 23 | Dictionaries - functions | mm/dd |  |  |
| 24 | Dictionaries - lists of dictionaries | mm/dd |  |  |
| 25 | Dictionaries - nested lists and dictionaries | mm/dd |  |  |

**Content**

The content of reading assignments isn't always just text; video and images can be used as well.

# Truth tables

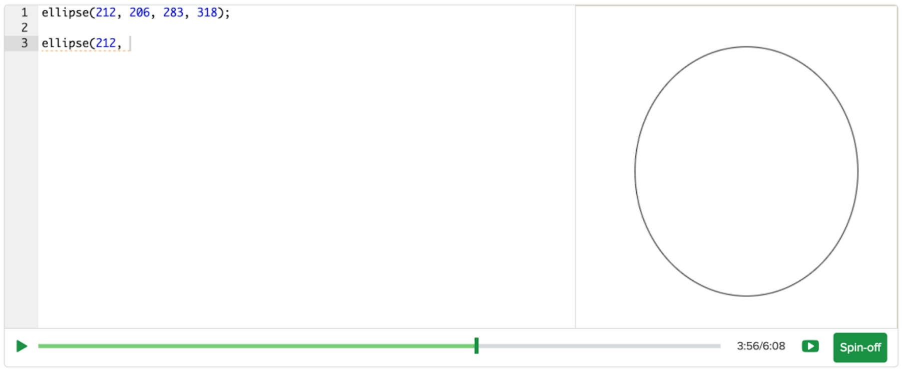
This chapter introduces a way of evaluating sentences and arguments of SL. Although it can be laborious, the truth table method is a purely mechanical procedure that requires no intuition or special insight.

## 3.1 Truth-functional connectives

Any non-atomic sentence of SL is composed of atomic sentences with sentential connectives. The truth-value of the compound sentence depends only on the truth-value of the atomic sentences that comprise it. In order to know the truth-value of $(D \leftrightarrow E)$, for instance, you only need to know the truth-value of $D$ and the truth-value of $E$. Connectives that work in this way are called TRUTH-FUNCTIONAL.

In this chapter, we will make use of the fact that all of the logical operators in SL are truth-functional— it makes it possible to construct truth tables to determine the logical features of sentences. You should realize, however, that this is not possible for all languages. In English, it is possible to form a new sentence from any simpler sentence $\mathcal{X}$ by saying 'It is possible that $\mathcal{X}$.' The truth-value of this new sentence does not depend directly on the truth-value of $\mathcal{X}$. Even if $\mathcal{X}$ is false, perhaps in some sense $\mathcal{X}$ *could* have been true— then the new sentence would be true. Some formal languages, called *modal logics*, have an operator for possibility. In a modal logic, we could translate 'It is possible that $\mathcal{X}$' as $\diamond\mathcal{X}$. However, the ability to translate sentences like these come at a cost: The $\diamond$ operator is not truth-functional, and so modal logics are not amenable to truth tables.

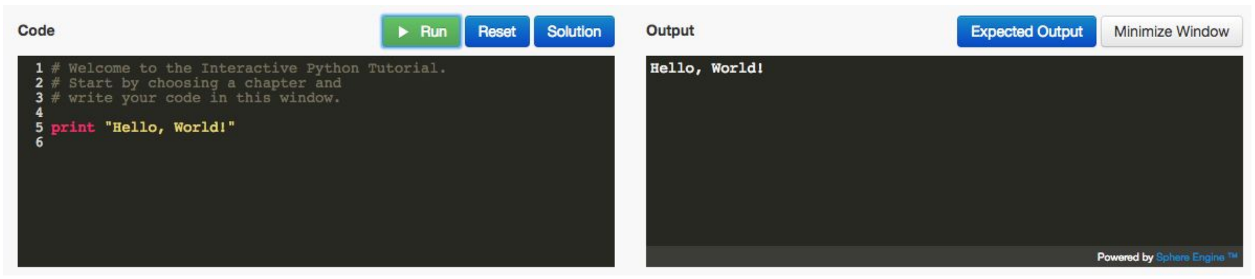
Making drawings with code



```
ellipse(212, 206, 283, 318);

ellipse(212,
```

## Interactive Coding Simulations

Throughout the content there may be several small coding simulations that users can tinker with to apply the knowledge gained in the text. They are not required, but they can give users useful feedback to help their coding skills develop.

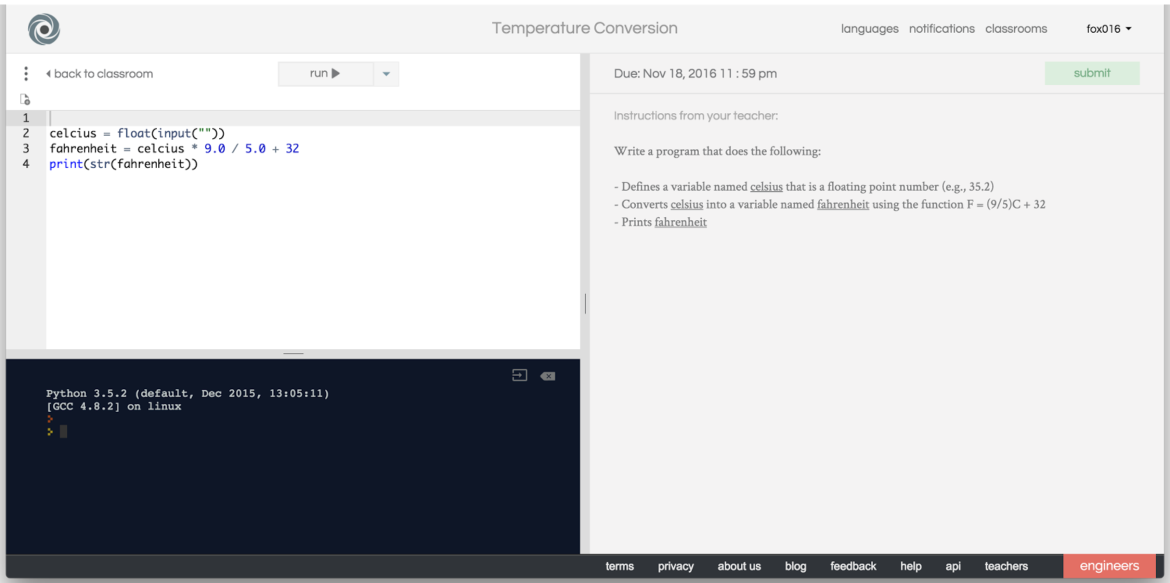


```
# Welcome to the Interactive Python Tutorial.
# Start by choosing a chapter and
# write your code in this window.

print "Hello, World!"

```

```
Hello, World!
```

## Assessment - Coding

There are assessments at the end of every reading assignment. Some questions may ask users to write some code to perform simple functions discussed in the reading. The interface will be similar to that for coding assignments: there will be an a description of the problem, an area to write and test code, and a submit button to click once the user thinks the code is working. The main difference between these assessments and coding assignments is that these assessments are generally much simpler, and if incorrect code is submitted then the user is directed back to the place in the reading that will help them.



Due: Nov 18, 2016 11 : 59 pm

Instructions from your teacher:

Write a program that does the following:

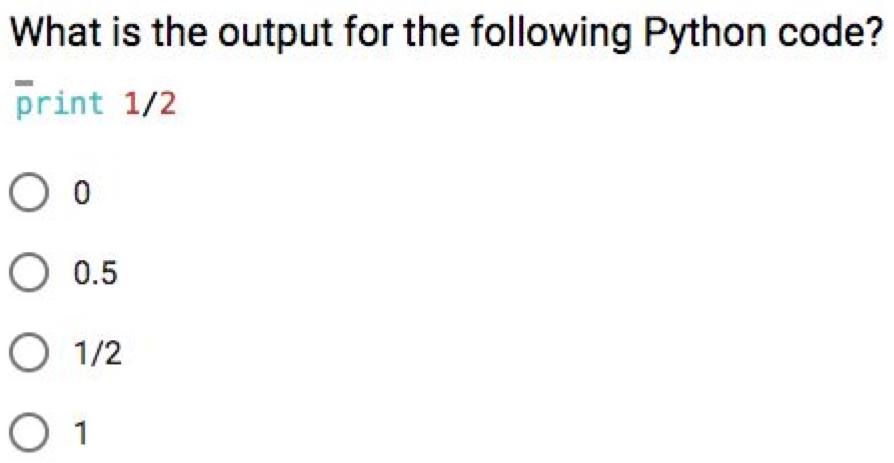
- Defines a variable named celsius that is a floating point number (e.g., 35.2)
- Converts celsius into a variable named fahrenheit using the function F = (9/5)C + 32
- Prints fahrenheit

```python

celcius = float(input(""))
fahrenheit = celcius * 9.0 / 5.0 + 32
print(str(fahrenheit))
```

```
Python 3.5.2 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
>
>
```

**Assessment - Multiple Choice**

Some reading assessment questions will be multiple choice. Users may be asked to select a block of code that produces some given output or to select the correct output given a block of code. Multiple choice questions may also be more traditional, asking the users to recognize the correct answer to a question from a list of options.



What is the output for the following Python code?

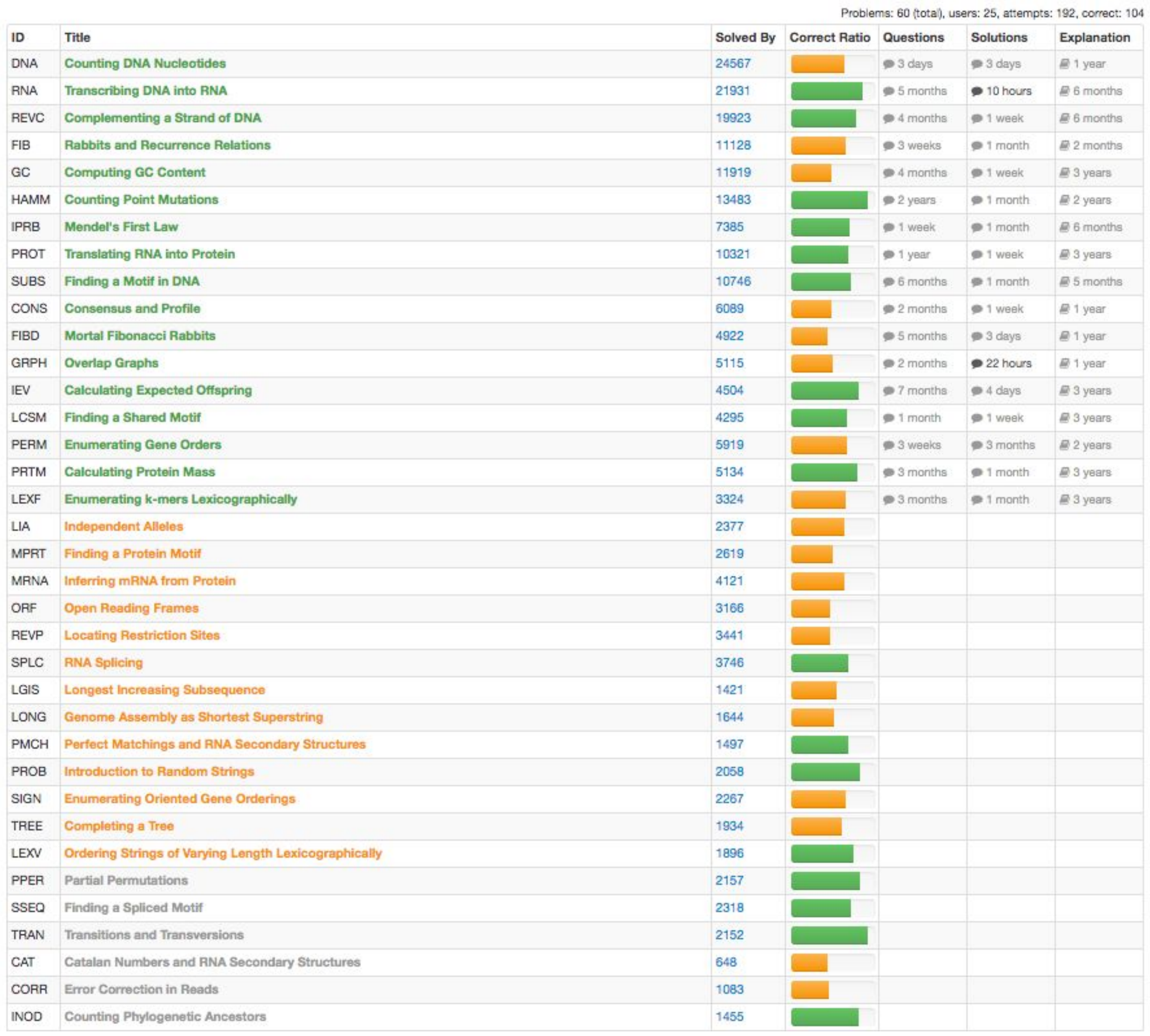
```python
print 1/2
```

- ○ 0
- ○ 0.5
- ○ 1/2
- ○ 1

## Coding Assignments

### List View

The list view shows a tabular view of all coding assignments. Assignments are green if they have been completed by the user, orange if all the prerequisite assignments have been completed by the user, or gray if they are not yet available to the user. All users can see the name of the assignment, the number of people in the course who have solved it, and the discussion board for the assignment. All users who have completed the assignment can also see their individual score and solutions submitted by the instructor and their other classmates. The list view does not show the prerequisites for an assignment.

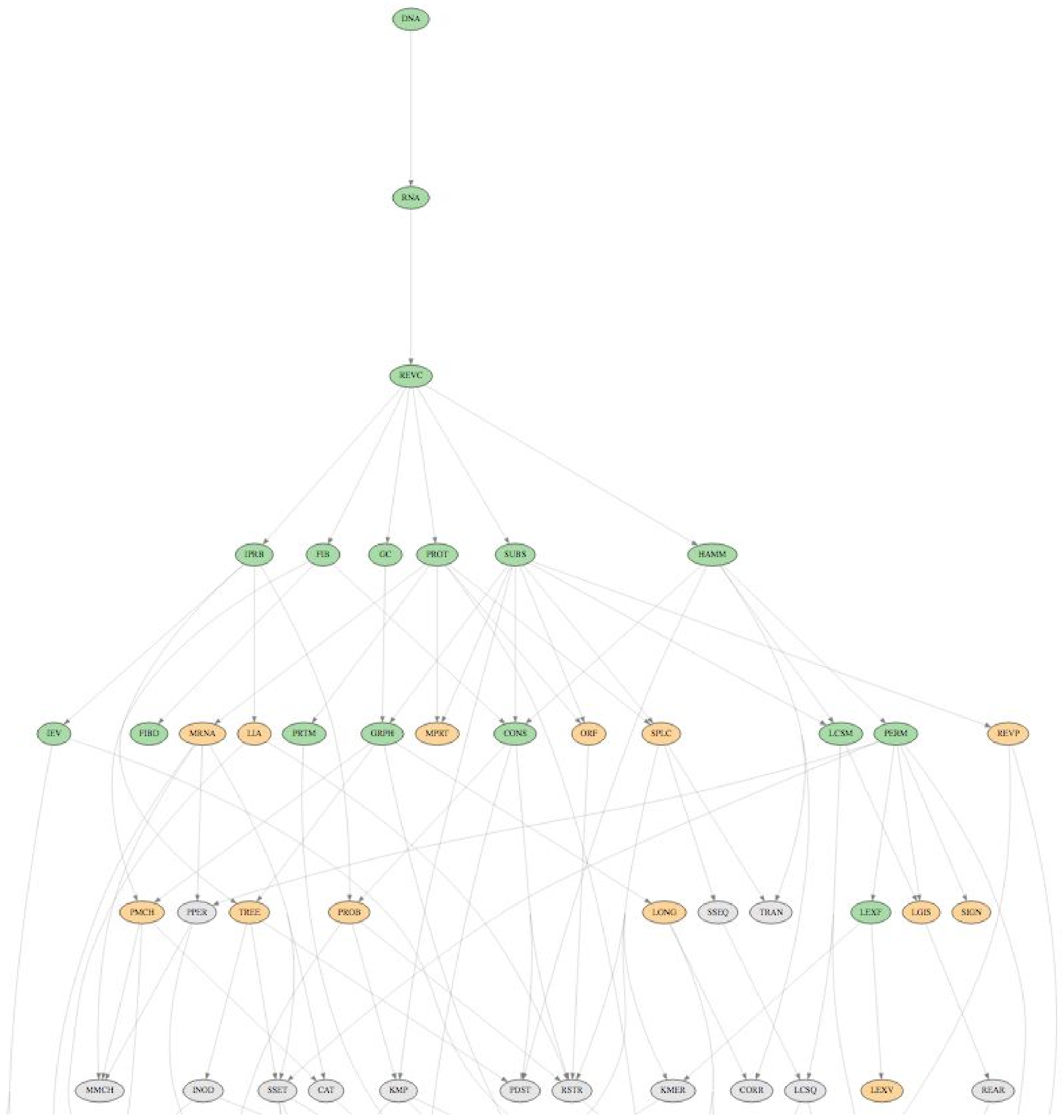


Problems: 60 (total), users: 25, attempts: 192, correct: 104

| ID | Title | Solved By | Correct Ratio | Questions | Solutions | Explanation |
|---|---|---|---|---|---|---|
| DNA | Counting DNA Nucleotides | 24567 | 🟧 | 3 days | 3 days | 1 year |
| RNA | Transcribing DNA into RNA | 21931 | 🟩 | 5 months | 10 hours | 6 months |
| REVC | Complementing a Strand of DNA | 19923 | 🟩 | 4 months | 1 week | 6 months |
| FIB | Rabbits and Recurrence Relations | 11128 | 🟧 | 3 weeks | 1 month | 2 months |
| GC | Computing GC Content | 11919 | 🟧 | 4 months | 1 week | 3 years |
| HAMM | Counting Point Mutations | 13483 | 🟩 | 2 years | 1 month | 2 years |
| IPRB | Mendel's First Law | 7385 | 🟩 | 1 week | 1 month | 6 months |
| PROT | Translating RNA into Protein | 10321 | 🟩 | 1 year | 1 week | 3 years |
| SUBS | Finding a Motif in DNA | 10746 | 🟩 | 6 months | 1 month | 5 months |
| CONS | Consensus and Profile | 6089 | 🟧 | 2 months | 1 week | 1 year |
| FIBD | Mortal Fibonacci Rabbits | 4922 | 🟧 | 5 months | 3 days | 1 year |
| GRPH | Overlap Graphs | 5115 | 🟧 | 2 months | 22 hours | 1 year |
| IEV | Calculating Expected Offspring | 4504 | 🟩 | 7 months | 4 days | 3 years |
| LCSM | Finding a Shared Motif | 4295 | 🟩 | 1 month | 1 week | 3 years |
| PERM | Enumerating Gene Orders | 5919 | 🟧 | 3 weeks | 3 months | 2 years |
| PRTM | Calculating Protein Mass | 5134 | 🟩 | 3 months | 1 month | 3 years |
| LEXF | Enumerating k-mers Lexicographically | 3324 | 🟧 | 3 months | 1 month | 3 years |
| LIA | Independent Alleles | 2377 | 🟧 | | | |
| MPRT | Finding a Protein Motif | 2619 | 🟧 | | | |
| MRNA | Inferring mRNA from Protein | 4121 | 🟧 | | | |
| ORF | Open Reading Frames | 3166 | 🟧 | | | |
| REVP | Locating Restriction Sites | 3441 | 🟧 | | | |
| SPLC | RNA Splicing | 3746 | 🟩 | | | |
| LGIS | Longest Increasing Subsequence | 1421 | 🟧 | | | |
| LONG | Genome Assembly as Shortest Superstring | 1644 | 🟧 | | | |
| PMCH | Perfect Matchings and RNA Secondary Structures | 1497 | 🟩 | | | |
| PROB | Introduction to Random Strings | 2058 | 🟩 | | | |
| SIGN | Enumerating Oriented Gene Orderings | 2267 | 🟧 | | | |
| TREE | Completing a Tree | 1934 | 🟧 | | | |
| LEXV | Ordering Strings of Varying Length Lexicographically | 1896 | 🟩 | | | |
| PPER | Partial Permutations | 2157 | 🟩 | | | |
| SSEQ | Finding a Spliced Motif | 2318 | 🟩 | | | |
| TRAN | Transitions and Transversions | 2152 | 🟩 | | | |
| CAT | Catalan Numbers and RNA Secondary Structures | 648 | 🟧 | | | |
| CORR | Error Correction in Reads | 1083 | 🟧 | | | |
| INOD | Counting Phylogenetic Ancestors | 1455 | 🟩 | | | |

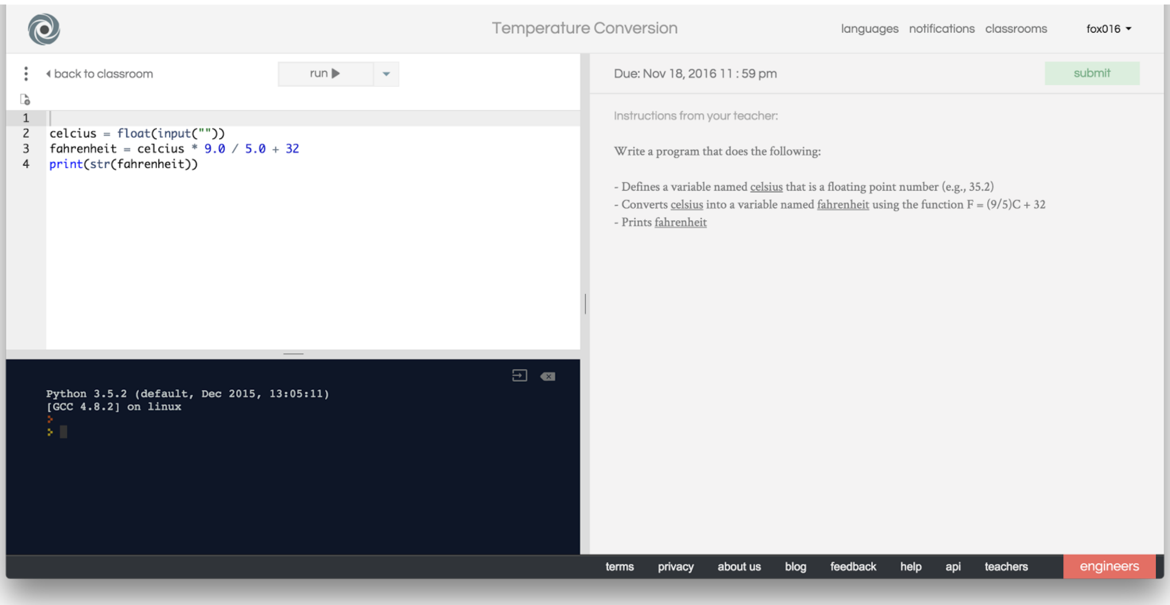
## Tree View

The tree view shows much less data than the tabular list view. Color codes are the same, and the only other information conveyed in the tree view is the prerequisites for each assignment.



## Assignment View

The assignment view has a description of the problem, a development environment to write code, a way to enter input and view output (for testing), and a button to submit code. Users can submit multiple times until the autograder passes it off as complete. Users can also choose to mark an assignment as a zero, which gives the user no credit but allows the user to see solutions submitted by the instructor and other users.



Due: Nov 18, 2016 11 : 59 pm

```

celcius = float(input(""))
fahrenheit = celcius * 9.0 / 5.0 + 32
print(str(fahrenheit))
```

```
Python 3.5.2 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
>
>
```

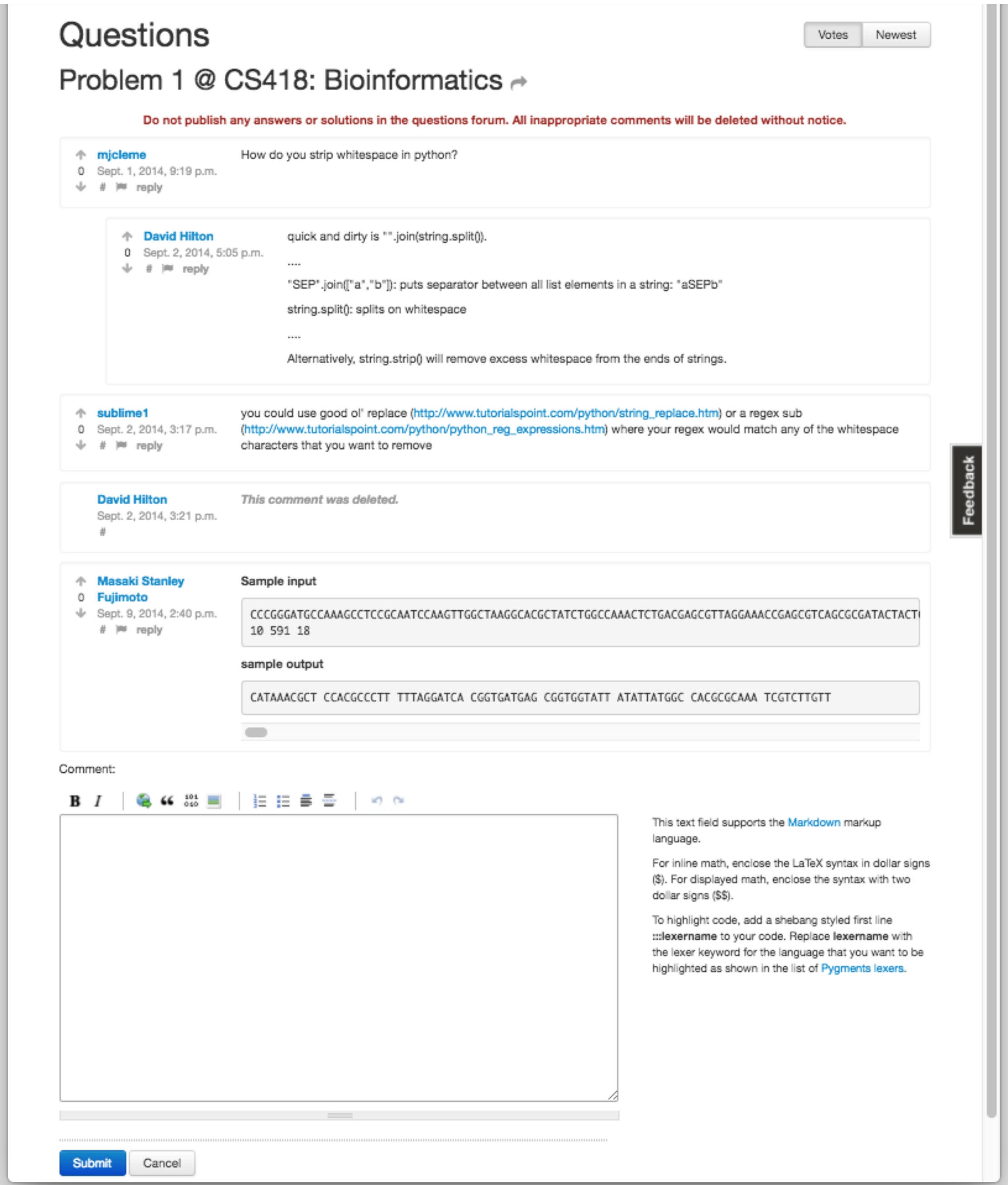
Instructions from your teacher:

Write a program that does the following:

- Defines a variable named celsius that is a floating point number (e.g., 35.2)
- Converts celsius into a variable named fahrenheit using the function F = (9/5)C + 32
- Prints fahrenheit

## Discussion Board

Each individual assignment has its own discussion board, and there is a general discussion board as well. Everyone in the class can see everything posted in the discussion boards; there is no way to only communicate to a subset of users through the online portal (students can use Learning Suite messages for that).



Votes Newest

# Questions

## Problem 1 @ CS418: Bioinformatics

**Do not publish any answers or solutions in the questions forum. All inappropriate comments will be deleted without notice.**

**mjcleme**
0 Sept. 1, 2014, 9:19 p.m.
# reply

How do you strip whitespace in python?

**David Hilton**
0 Sept. 2, 2014, 5:05 p.m.
# reply

quick and dirty is "".join(string.split()).

....

"SEP".join(["a","b"]): puts separator between all list elements in a string: "aSEPb"

string.split(): splits on whitespace

....

Alternatively, string.strip() will remove excess whitespace from the ends of strings.

**sublime1**
0 Sept. 2, 2014, 3:17 p.m.
# reply

you could use good ol' replace (http://www.tutorialspoint.com/python/string_replace.htm) or a regex sub (http://www.tutorialspoint.com/python/python_reg_expressions.htm) where your regex would match any of the whitespace characters that you want to remove

**David Hilton**
Sept. 2, 2014, 3:21 p.m.
#

*This comment was deleted.*

**Masaki Stanley Fujimoto**
0 Sept. 9, 2014, 2:40 p.m.
# reply

**Sample input**

```
CCCGGGATGCCAAAGCCTCCGCAATCCAAGTTGGCTAAGGCACGCTATCTGGCCAAACTCTGACGAGCGTTAGGAAACCGAGCGTCAGCGCGATACTACT
10 591 18
```

**sample output**

```
CATAAACGCT CCACGCCCTT TTTAGGATCA CGGTGATGAG CGGTGGTATT ATATTATGGC CACGCGCAAA TCGTCTTGTT
```

Comment:

This text field supports the Markdown markup language.

For inline math, enclose the LaTeX syntax in dollar signs ($). For displayed math, enclose the syntax with two dollar signs ($$).

To highlight code, add a shebang styled first line **:::lexername** to your code. Replace **lexername** with the lexer keyword for the language that you want to be highlighted as shown in the list of Pygments lexers.

Sample Assignments

## Sample Reading Assignment

Numbers

Python supports two types of numbers—integers and floating point numbers. An integer can be referred to as an “int” for short, and a floating point number can be referred to as a “float”. An integer is a whole number, which means it has no fraction or decimal part.

To define an integer, use the following syntax:

```
myint = 7
```

To define a floating point number, you may use one of the following notations:

```
myfloat = 7.0
myfloat = float(7)
```

Coding Simulation: Create a variable named myint that has a value of 3. Create another variable named myfloat that has a value of 3.0. Use Python’s print command to print the variable values. Then, use the print command and Python’s type function to print the types, like this:

```
print myint
print myfloat
print type(myint)
print type(myfloat)
```

You should see the following output:

```
3
3.0
<type 'int'>
<type 'float'>
```

In Python, you can perform mathematical operations on ints and floats. Operations can be performed on the numbers themselves (e.g. print 3 + 3.0) or on the variable names (e.g. print myint + myfloat).

Common operations are:

| | |
|---|---|
| + | Addition `print 1 + 1` `2` |
| - | Subtraction `print 6 - 2` `4` |
| * | Multiplication `print 5 * 2` `10` |
| / | Division `print 4 / 2` `2` `print 14 / 4` `3` (both numbers are ints, so the result is an int rounded down) `print 14 / 4.0` `3.5` |
| ** | Exponent `print 2**5` `32` |
| % | Modulo (takes the remainder after division) `print 14 % 4` `2` (because 14 divided by 4 is 3 remainder 2) |

Coding Simulation: What is the resulting data type if you add 2 ints together? 2 floats together? An int and a float? What is the resulting data type if you divide 2 ints (e.g., “print 7/8" and “print type(7/8)”)? Knowing what you do about ints and floats, why would this be?

Strings

Strings contain textual data. You can define a string by enclosing text in single quotes (‘’) or double quotes(“”):

```
mystring = “Hello, world!”
mystring = ‘Hello, world!’
```

You can add strings together to concatenate them:

```
greeting = “Hello, ”
name = “Bob”
print greeting + name + “!”
```

Coding Simulation: Experiment with making strings and adding them together. Using at least three different variables, try printing, “Susie told Becky that Alyssa is her new best friend!”

What if you want to print a number and a string? If you try to add a number and a string, Python will throw an error. There are two ways around this. One, you can turn the number into a string using the str() function:

```
age = 10
name = “Bob”
print name + “ is ” + str(age) + “ years old!”
```

The other way is to print a comma-separated list of strings, numbers, and variables. When the print command sees a list of data to print, Python automatically converts all the data to strings and separates each resulting string with spaces. For example:

```
age = 10
name = "Bob"
print name, "is", age, "years old!"
```

In this case, it will still print “Bob is 10 years old!” It will convert the age variable to a string, then print out the list of strings with spaces separating them.

Input From Keyboard

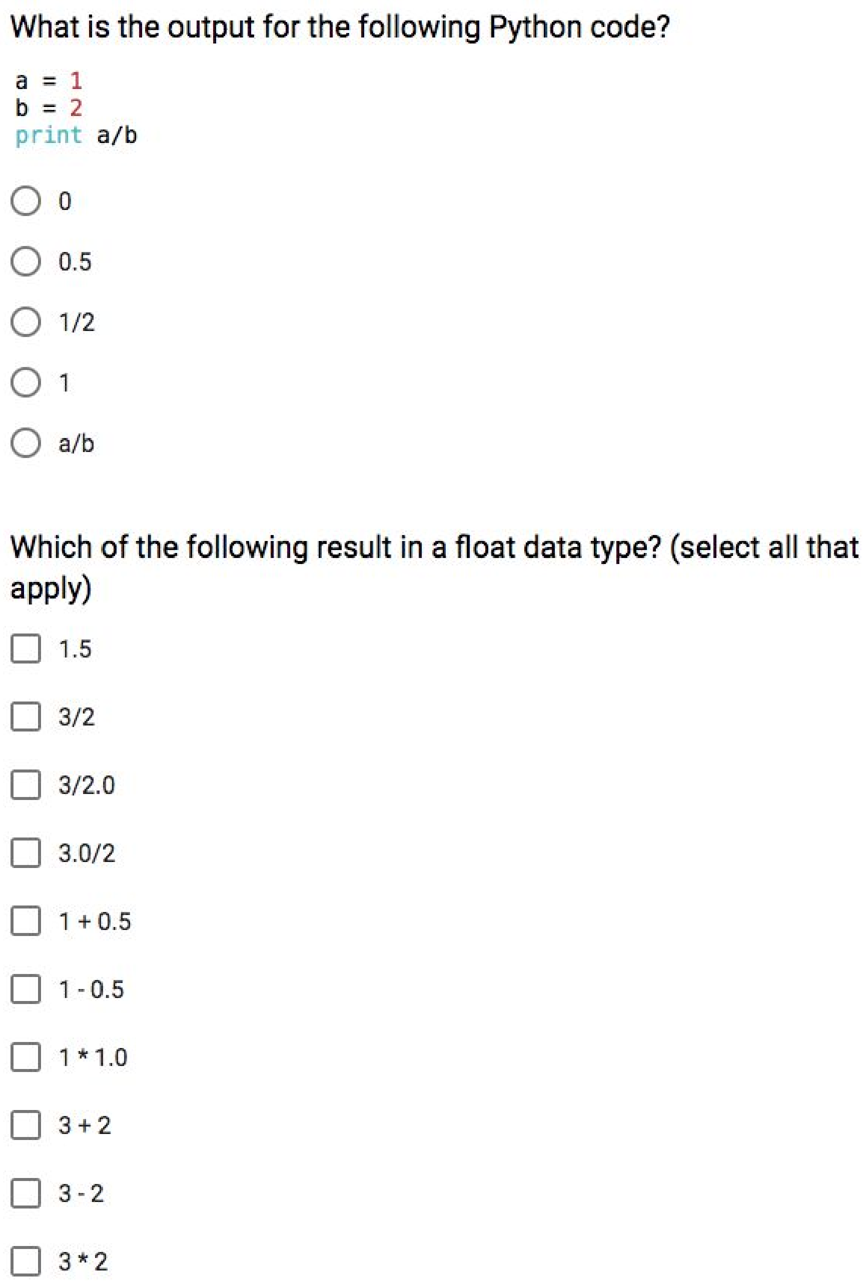
To get input from the keyboard, use the raw_input() function. The raw_input() function will display a string to the user, pause and wait for the user to type in input, then once the user hits “enter” the input will be stored as a string. For example:

```
name = raw_input("Enter your name: ")
print "Your name is " + name
```

But what if you want the keyboard input to be a number? What if you want users to input their age so your code can tell them how old they’ll be in ten years? What if you want users to input a dollar amount or an interest rate for a program to calculate how much money they’re making over time? You can use the int() or float() functions to convert keyboard input into a number data type:

```
age = int(raw_input("Enter your age: ")
print "You will be " + str(age+10) " years old in 10 years!"

principal = float(raw_input("Enter principal amount: "))
rate = float(raw_input("Enter a growth rate: "))
time = int(raw_input("Enter number of years: "))
print "Your simple interest earned will be $", p*r*t
```

## Sample Reading Assessment



What is the output for the following Python code?

```
a = 1
b = 2
print a/b
```

○ 0

○ 0.5

○ 1/2

○ 1

○ a/b

Which of the following result in a float data type? (select all that apply)

☐ 1.5

☐ 3/2

☐ 3/2.0

☐ 3.0/2

☐ 1 + 0.5

☐ 1 - 0.5

☐ 1 * 1.0

☐ 3 + 2

☐ 3 - 2

☐ 3 * 2