**Introduction to Computer Science (CS) Syllabus**

**Class Mission Statement:**

We strive to develop a foundation of **computer science** knowledge and learn new approaches to problem solving that harness the power of computational thinking to become both users and creators of **computing** technology………

**Course Description:**

The intro course uses Snap! a blocked-based programming environment and transitions to Python, a text-based programming language.

[**TEALS Intro to CS Semester 1: Snap!**](https://tealsk12.github.io/introduction-to-computer-science/)[**TEALS Intro to CS Semester 2: Python**](https://tealsk12.github.io/2nd-semester-introduction-to-computer-science/)

Computing has changed the world in profound ways: it has opened wonderful new ways for people to connect, design, research, play, create, and express themselves. However, using the computer is just a small part. This course is an introductory programming course that helps prepare students for more advanced programming courses. The Intro to Computer Science uses Snap!. an approachable, visual, block-based programming language, with a robust tool set, perfect for introducing students to code for the first time. Snap! is taught as a single semester course. Then students will benefit from exposure to a Python-based programming curriculum.

**Prerequisites**

**Technology Requirements**

A desktop or laptop computer must be made available to each student for use during class. For the first semester students will need access to the Snap! programming environment, available at [**http://snap.berkeley.edu/**](http://snap.berkeley.edu/).

For the python curriculum, students will need access to a python development environment. [**Microsoft’s Visual Studio Code**](https://code.visualstudio.com/docs/languages/python) is recommended for students with Windows, Mac, or Linux computers.

**Required Materials:**

Notebook

Pencils / Pens

Headphones or Earbuds

**Curriculum Plan**

**Semester 1**

**Unit 0: Beginnings**

|  |  |
| --- | --- |
| Lesson | Objectives |
| 0.1: The First Day | Identify the class they are taking. List the high-level goals of the course. Describe classroom procedures, rules, and norms. |
| 0.2: Algorithms | Define “algorithm.” Construct algorithms for performing simple tasks. |
| 0.3: Programming Languages | Complete levels in the game LightBot 2.0. Complete small programs in SNAP with guidance. Explain why computer programs are written in specialized languages. |
| 0.4: Snap Self Portrait | Create a simple “program” in SNAP to describe themselves |
| 0.5: Snap Coordinate System | Create a drawing using the Snap! Coordinate System |

**Unit 1: SNAP Basics**

|  |  |
| --- | --- |
| **Lesson** | **Objectives** |
| [1.1: Welcome to SNAP](https://tealsk12.github.io/introduction-to-computer-science/lesson_11.md.html) | Define and identify “blocks,” “scripts,” “sprites,” and “the stage” in SNAP. Write simple SNAP programs. Describe what simple SNAP programs do without executing the code. |
| [1.2: Building Blocks](https://tealsk12.github.io/introduction-to-computer-science/lesson_12.md.html) | Name the categories of blocks in SNAP and describe what the blocks in each category do. Describe the function of several common SNAP blocks. Be able to use common blocks to build simple SNAP programs. |
| [1.3: Drawing Shapes](https://tealsk12.github.io/introduction-to-computer-science/lesson_13.md.html) | Construct simple algorithms to draw shapes. Convert algorithms into SNAP programs. |
| [1.4: Animation](https://tealsk12.github.io/introduction-to-computer-science/lesson_14.md.html) | Animate SNAP sprites using costume changes and movement. Trigger action in other sprites using broadcasts. |
| 1.5: Storytelling Project | Apply basic programming and SNAP skills to create an animated movie, play, nursery rhyme, or other scene. Practice good debugging skills to correct issues as they arise while programming |

**Unit 2: Loop-de-Loop**

|  |  |
| --- | --- |
| **Lesson** | **Objectives** |
| 2.1: Loops | Define “loop” in a programming context. Explain why loops are useful. Implement simple repeat and forever loops in SNAP. Utilize loops to reduce redundancy in code. |
| 2.2: Nested Loops | Use nested loops to solve programming problems. |
| 2.3: Inputs and Conditionals | Ask for and receive user input in a SNAP program. Use simple conditional (if and if-else) blocks to alter control flow in a SNAP program. |
| 2.4: Variables | Use variables to track values throughout a program. |
| 2.5: Boole in the House | Define and identify Boolean expressions and operators. Evaluate Boolean expressions. Utilize Boolean operators (and/or/not) to create compound conditions. |
| 2.6: Pong Project | Implement a well-written version of Pong. Practice good style and conventions to create readable and maintainable code. |

**Unit 3: Variables and Customization**

|  |  |
| --- | --- |
| **Lesson** | **Objectives** |
| 3.1: Abstraction and Generalization | Define abstraction, detail removal, generalization, and procedural decomposition in a computer science context. Describe how utilizing procedural decomposition can improve the readability and maintainability of algorithms and/or code. Recognize opportunities to improve algorithms by abstracting or generalizing parts into sub procedures. |
| 3.2: Combining Loops and Conditionals | Build custom command blocks in Snap. Utilize detail removal and generalization to construct blocks that practice abstraction. |
| 3.3: Customization I: Arguments | Build custom SNAP blocks that take arguments. |
| 3.4: Customization II: Reporters and Predicates | Build custom reporter and predicate blocks in SNAP. |
| 3.5: Platform Game Project | Use loops, variables, and Boolean expressions to implement a Super Mario Bros. style platform game. Practice good debugging skills to correct issues as they arise while programming. |

**Unit 4: Lists**

|  |  |
| --- | --- |
| **Lesson** | **Objectives** |
| 4.1: Intro to Lists | Explain the concept of a “list” in a programming context. Identify scenarios in which lists are useful |
| 4.2: Static Lists | Create static lists in SNAP. Access elements of a list. Add and remove elements from a list. |
| 4.3: List Practice I | Traverse a list, accessing each element one at a time. Perform operations combining all elements in a list. Select defined subsets of elements in a list. |
| 4.4: List Practice II | Traverse a list, accessing each element one at a time. Perform operations combining all elements in a list. Select defined subsets of elements in a list. |
| 4.5: Sequential Search | Explain the sequential search algorithm. Implement several variations of sequential search. |
| 4.6: Word Guessing Game Project | Use lists to implement a complete version of a word guessing game. Exercise good programming practices to produce code that is not only functional but also elegant and well-written. |

**Unit 5: Cloning**

|  |  |
| --- | --- |
| **Lesson** | **Objectives** |
| 5.1: Intro to Cloning | Explain why prototyping and clones can be useful. Describe how complex goals can be accomplished using cloning. |
| 5.2: Cloning Sprites | Demonstrate the difference between sprite and global variables. Explain how cloning and prototyping simplify working with numerous similar sprites in the same program. Create prototype sprites and clones of the prototype sprite. Explain the difference between a “master” sprite and a “clone” sprite. |
| 5.3: Communicating with Clones | Pass information to individual clones. [Optional] Describe a race condition that might occur due using global variables and clones. Delete clones when they are no longer needed. |
| 5.4: Space Invaders | Use cloning to implement a complete version of “Space Invaders.” Exercise good programming practices to produce code that is not only functional but also elegant and well-written. |

**Unit 6: Final Project**

|  |  |
| --- | --- |
| Lesson | Objectives |
| 6.1: Design Basics | Identify the key considerations when designing a piece of software. Describe methods for prioritizing features, use cases, and/or scenarios. Explain why design and planning are necessary steps in the software engineering process |
| 6.2: Brainstorming and Evaluating | Identify factors to use when choosing between project ideas. Rank a group of proposed project ideas using the identified factors |
| 6.3: Spec Writing | Identify the main components of a functional project specification and explain the purpose of each section. Develop a project idea into a full, detailed specification |
| 6.4: Building a Plan | Identify the main components of a functional project specification and explain the purpose of each section. Develop a project idea into a full, detailed specification. |
| 6.5: Project Implementation | Use the skills developed throughout the course to implement a medium- to large-scale software project. Realistically evaluate progress during software development and identify when cuts are necessary. Prioritize features and scenarios and choose which should be eliminated or modified if/when resources and/or time become limited. |

**Unit 1 Map - Introduction to Python**

|  |  |
| --- | --- |
| **Lesson** | **Objectives** |
| 1.01: Set Up | Define and identify: IDE, Python. Identify the key concepts that will be covered in the course. Set up and log into an account for the course's online IDE. Save and turn in a file via the online IDE. |
| 1.02: Interactive Mode | Define and identify interpreter, string, integer, float, value, errors, console, expression. Use the Python interpreter to evaluate simple math expressions. Distinguish between an integer, float, and string. |
| 1.03: Script Mode and Variables | Define and identify script, print, run, output, variable. Write a simple script and run it in the IDE. Print values out to the console (both composed values and from variables). Compare script mode vs interactive mode. Know how to store a value into a variable. |
| 1.04: Variables Input | Define and identify comments, storing, mutability, variable assignment, input. Assign and swap variables. Store user input into a variable. |
| 1.05: Quiz & Debugging | Demonstrate their understanding of key concepts covered up to this point. Define and identify debugging, syntax errors. Analyze and respond to error messages. |
| 1.06: Project | Apply basic Python knowledge about inputs/outputs and variables to create a game of Madlibs. Practice good debugging skills. |

**Unit 2 Map - Data Types and Conditionals**

|  |  |
| --- | --- |
| **Lesson** | **Objectives** |
| 2.01: Data Types & Casting | Define and identify type, string, casting, floating point number (float), integer. Describe different representations of data in Python. Convert from one data type to another data type. |
| 2.02: Booleans & Expressions | Define and identify Boolean, expression, composition, True, False. Evaluate a Boolean expression. Compose Boolean expressions using and, or, not, <, >, and ==. |
| 2.03: Conditionals | Define and identify: if, else, elif, conditionals, flow of control. Create chaining if statements. Understand how conditional statements alter the flow of control of a program. |
| 2.04: Lists | Define and identify list, item, index, integer. Be able to access items from a list using the index. Create lists of different types. Use the length function. |
| 2.05: Lists 2 | Define and identify slice, append, pop, remove. Slice a list. Add and remove elements from a list |
| 2.06: Game Loop | Define and identify while loop. Use a while loop to simulate game play. |
| 2.07: Project | Use knowledge of lists, Booleans, conditionals, and while loops to create a text-based adventure game. |

**Unit 3 Map - Functions**

|  |  |
| --- | --- |
| **Lesson** | **Objectives** |
| 3.01: Built In Functions | Define and identify function, arguments, calling, importing, returning. Call the built-in randint function, using arguments. Utilize code other people have written in the Python documentation. Understand the difference between printing and returning. |
| 3.02: User-Defined Functions | Define and identify abstraction, def. Create functions. |
| 3.03: Return vs Print | Define and identify: return, none, void. Explain and demonstrate the difference between printing and returning. |
| 3.04: Debugging and Scope | Define and identify scope, aliasing, stack trace. Understand that changing a list in a function updates the list outside of the function. Understand that updating variables in a function does not affect the variable outside of the function. Understand global variables. Draw a simple stack trace. |
| 3.05: Project | Use project planning skills to complete a longer-term project. Create functions to organize a project. Apply skills learned in units 1-3 to create a functioning program. |

**Unit 4 Map - Nested Loops and Lists**

|  |  |
| --- | --- |
| **Lesson** | **Objectives** |
| 4.01: Looping Basics | Define and identify for loop, item, iteration, scope. Recall looping in Snap! and reapply the concept in Python. Loop through (traverse) the items in a list. Be aware of the scope of variables during iteration. |
| 4.02: For Loops | Define and identify range. Use the range and len() function to update lists via for loops. |
| 4.03: Nested For Loops | Define and identify nested for loops, stack trace. Use nested for loops via a function and a for loop. Use nested for loops via two loops nested. Use a stack trace to understand and demonstrate the flow of nested for loops. |
| 4.04: Nested Lists & Looping | Define and identify nested list. Use nested for loops to traverse through nested lists. |
| 4.05: Debugging and Quiz | Read and understand longer programs involving loops. Demonstrate knowledge of looping, lists, and nested loops/lists. Debug programs involving for loops and lists. |
| 4.06: Project | Use project planning skills to complete a larger project. Utilize loops, lists, and nested loops/lists to create a Tic-Tac-Toe game. |

**Unit 5 Map (Optional) - Music Programming**

|  |  |
| --- | --- |
| **Lesson** | **Objectives** |
| 5.01: Earsketch Intro | Define and identify: Digital Audio Workstation (DAW), sound tab, fitMedia(), setTempo(). Play beats using the above functions. Loop through items in a list. Be aware of the scope of variables during iteration. |
| 5.02: EarSketch Music | Define and identify: rhythm, beat, tempo, measures, setEffect(), makeBeat(). Play beats using the functions. Loop through items in a list. Be aware of the scope of variables during iteration. |
| 5.03: Earsketch Control Flow | Define and identify modulo. Review looping and control structures. Use looping concepts in music making via EarSketch. Use control structures to create music. |
| 5.04: EarSketch User-Defined Functions | Define and identify abstraction, section, A-B-A form. Create and apply user-defined functions to create songs with complicated form. |
| 5.05: Project | Create a complete song in EarSketch with multiple parts. Utilize EarSketch's features and functions. |

**Unit 6 Map - Dictionaries**

|  |  |
| --- | --- |
| **Lesson** | **Objectives** |
| 6.01: Introduction to Dictionaries | Define and identify dictionary, key, value. Create dictionaries of key-value pairs. Access and update items from dictionaries. |
| 6.02: Dictionaries Methods | Define and identify pop, default value. Update values in a dictionary. Add values to a dictionary. Remove values from a dictionary. |
| 6.03: Dictionaries of Lists | Create dictionaries with keys and values of different types. Update, append, or remove list values in a dictionary. |
| 6.04: Dictionaries Looping | Use loops to traverse through key/value pairs in a dictionary |
| 6.05: Project | Use dictionaries to create the game Guess Who |

[**Unit 7 Map - Introduction to Object Oriented Programming**](https://tealsk12.github.io/2nd-semester-introduction-to-computer-science/units/7_unit/unit7.md.html)

|  |  |
| --- | --- |
| **Lesson** | **Objectives** |
| 7.01: User-Defined Types | Define and identify class, instance, object, attributes. Create a class and instantiate. attributes to an instance. Manipulate instances and attributes through a function. |
| 7.02: User-Defined Types, Part 2 | Define and identify self, \_\_init\_\_. Create a class with an \_\_init\_\_ method. Understand and use the self-argument. Instantiate a class with arguments. |
| 7.03: Methods | Define and identify method, \_\_str\_\_, \_\_add\_\_, operator overloading. Create a class with an \_\_init\_\_ method. Understand and use the self-argument. Instantiate a class with an argument. |
| 7.04: Inheritance | Define and identify inheritance, parent class, child class. Create a class that inherits from anther class. Overwrite methods of parent class in a child class. |
| 7.05: Project | Engage in class design before beginning coding. Apply what was learned with respect to classes, methods, and inheritance to create an implementation of Pokemon. |

**Unit 8 Map - Final Project**

|  |  |
| --- | --- |
| Lesson | Objectives |
| 8.01: Final Project Brainstorming and Evaluating | Recall project planning basics from last semester. Identify factors to use when choosing between project ideas. Rank a group of proposed project ideas using the identified factors. |
| 8.02: Defining Requirements | Define key scenarios for a project and the features required to implement each scenario. Explain the importance of wireframing when designing an application. |
| 8.03: Building a Plan | Identify the main components of a functional project specification and explain the purpose of each section. Develop a project idea into a full, detailed specification. |
| 8.04: Project Implementation | Use the skills developed throughout the course to implement a medium- to large-scale software project. Realistically evaluate progress during software development and identify when cuts are necessary. Prioritize features and scenarios and choose which should be eliminated or modified if/when resources and/or time become limited. |

**Classroom Procedures**

**Expectations:**

**Do Now**: The Do Now is to be completed within the first 3-5 minutes of class in your notebook or online (see instructions for that day). You should begin it immediately and work on it silently (unless instructed otherwise). Your notebooks and/or binders will be collected and graded toward your class participation grade.

**Notebooks**

Taking notes on paper results in better learning outcomes for students. In computer science, notebooks can be used to record definitions, syntax, programming patterns and idioms, examples, and diagrams. Students can also reflect on the work they are doing and use the notebook as a scratch space to plan out their approach to problems before implementation. Notebooks help make learning more explicit and help students to organize and process new information.

* When should students use their notebooks in the class?
* How often will the team check the notebooks? Who checks them and when? Is there a grade associated with notebook completeness?
* When can and should students refer to their notes (during lab? On quizzes? On tests?)
* What should students do about class notes when they miss class?

**Notebook Checks:** It is extremely important that you keep all your course materials for the entire semester. You will have a Binder and Notebook check from time to time during the semester. All in-class assignments are “fair game” as well as all Do Now that we have done (including Do Now that took place when you were absent).