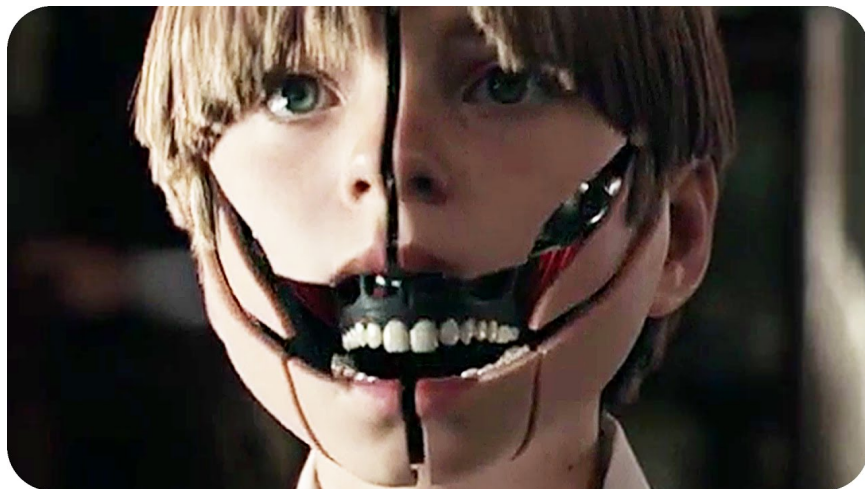


Introduction to Snap!

Game and Robotics Programming with Snap! and Python














August 22, 2023

Introduction



- Snap! vs. Scratch
- Scratch interface vs. Snap! interface
- Programming - why? how? what?
- Programming languages vs. natural languages
- Look under the hood: infrastructure you're (not) dealing with
- Why Snap! (Why not just Python?)

Snap! vs. Scratch

Jun 2023	Jun 2022	Change	Programming Language	Ratings	Change
1	1		 Python	12.46%	+0.26%
2	2		 C	12.37%	+0.46%
3	4	▲	 C++	11.36%	+1.73%
4	3	▼	 Java	11.28%	+0.81%
5	5		 C#	6.71%	+0.59%
6	6		 Visual Basic	3.34%	-2.08%
7	7		 JavaScript	2.82%	+0.73%
8	13	▲	 PHP	1.74%	+0.49%
9	8	▼	 SQL	1.47%	-0.47%
10	9	▼	 Assembly language	1.29%	-0.56%
11	12	▲	 Delphi/Object Pascal	1.26%	-0.07%
12	24	▲	 MATLAB	1.11%	+0.48%
13	25	▲	 Scratch	1.02%	+0.43%

- Web-based online application (HTML5), no desktop app though you can (and should) download it from here:
 1. In your browser, go to github.com/jmoenig/Snap/releases
 2. Click on "Source code (zip)" for instant download
 3. In your browser (or on your PC) navigate to **Downloads**
 4. **Extract** all files to any location (**Downloads** is OK)
 5. Go to the Snap directory, open it and click on **snap.html** to start
 6. Close the editor again, and enter **Snap** in the Windows search bar
 7. Pin the Snap! app to the taskbar for later use.

- Comparison between Snap! and Scratch (About Snap!):
 - Snap! allows the user to define **data structures** like trees, heaps, hash tables etc. because you can define a "list of lists".
 - In Snap!, you can also create **control structures** like functions that allow you to automate things.
 - Data and control structures are the backbone of any form of **imperative** computer programming.
- This enables you to do things you cannot do in Scratch, including anything other high-level languages like C++ or Python can do
- See also: Harvey & Mönig (2010). Bringing “No Ceiling” to Scratch: Can One Language Serve Kids and Computer Scientists? URL: berkeley.edu

Scratch interface

You can program the Sphero Bolt robot with Scratch (see [here](#)) or with Python (we’ll do that).

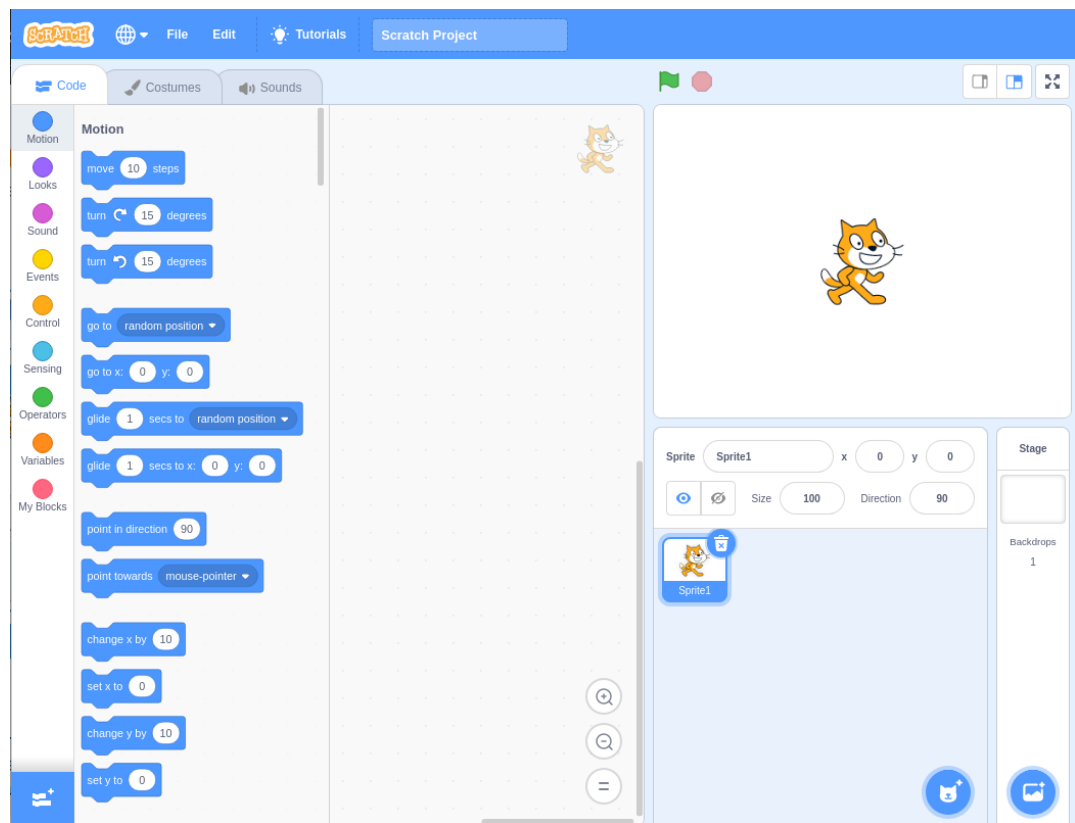


Figure 1: Scratch 3 interface



Figure 2: Sphero Bolt in human hand

Snap! interface and examples

Example: Conway's Game of Life

Basic Rules of Conway's Game of Life

1. Living cells die if they have fewer than 2 neighbors (underpopulation/loneliness)

2. Living cells die if they have more than 3 neighbors (overpopulation)

3. Dead cells that have 3 neighbors become alive (reproduction)

4. Otherwise, there is no change (whether cell is alive or dead)

5

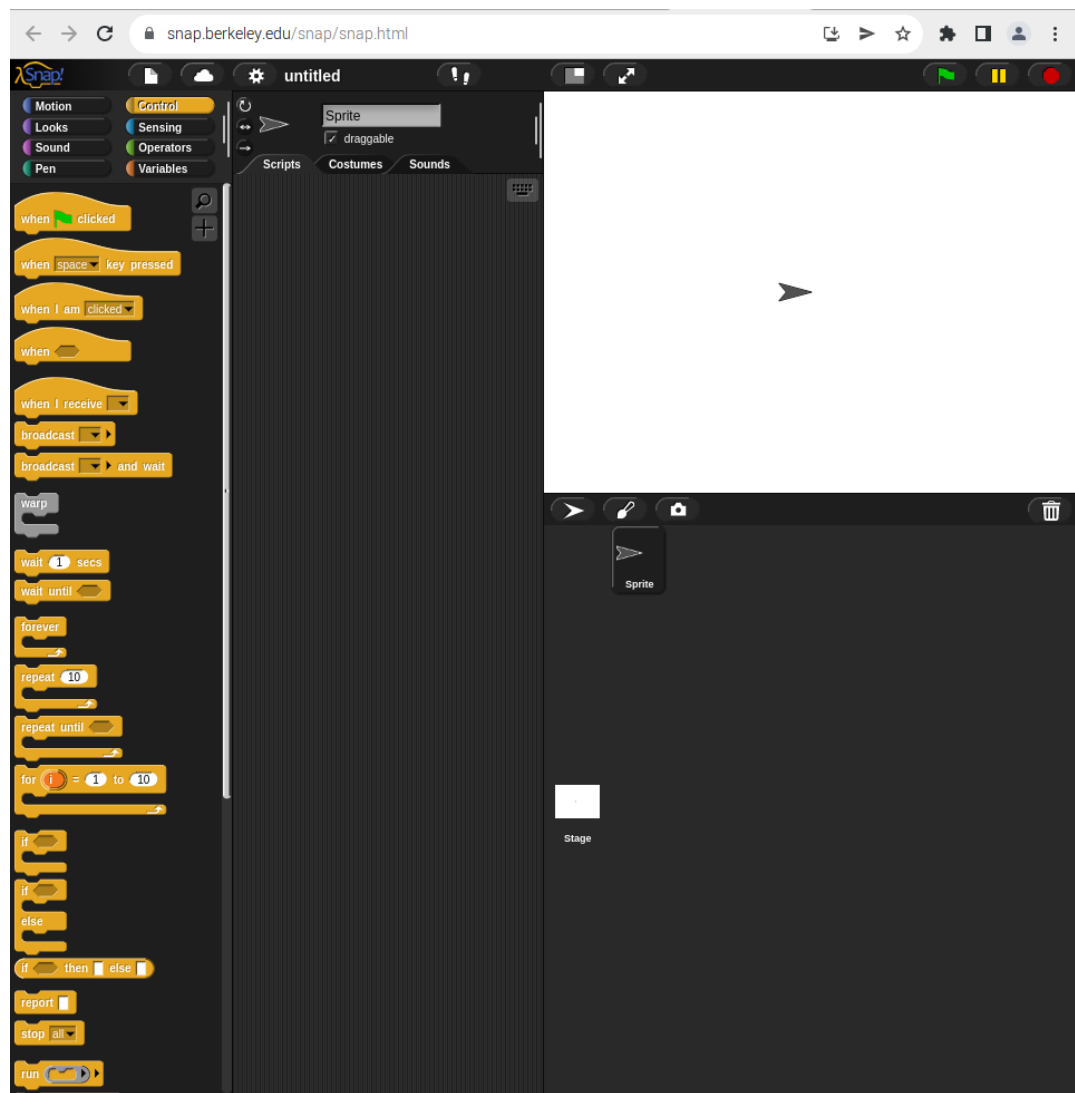


Figure 3: Snap! 7 interface

- Example: Conway's Game of Life - check it out now:
 1. In your browser, navigate to `snap.berkeley.edu`
 2. In the search bar, enter Conway
 3. Select the project Conway's Life infinite playing field
 4. Click on the "full screen symbol"
 5. Click on the **green flag**
 6. Enter cellular automata by clicking on the squares
 7. Start the simulation and let it run until the end
 8. Leave the full screen
 9. Select **See Code**
 10. Check out the program (we might get back to it)
- Check out one of my own creations: a "Time Machine" (of sorts)
 1. Go back to the Snap! home page at `snap.berkeley.edu`
 2. Search for **birkenkrahe**
 3. On my **public page** click **follow this user**
 4. Open the "Time Machine" project
 5. Go to "full screen"
 6. Click the **green flag**
 7. Click on the grey circle and move it around to see what happens
 8. Leave the full screen
 9. Click on **see code**
 10. Check out the code

Programming

- This could be your first ever programming course. Is it?
 - What were your other courses about?
 - What did you take away from them?
 - What's your view towards programming?
- Why should you bother to learn how to program?

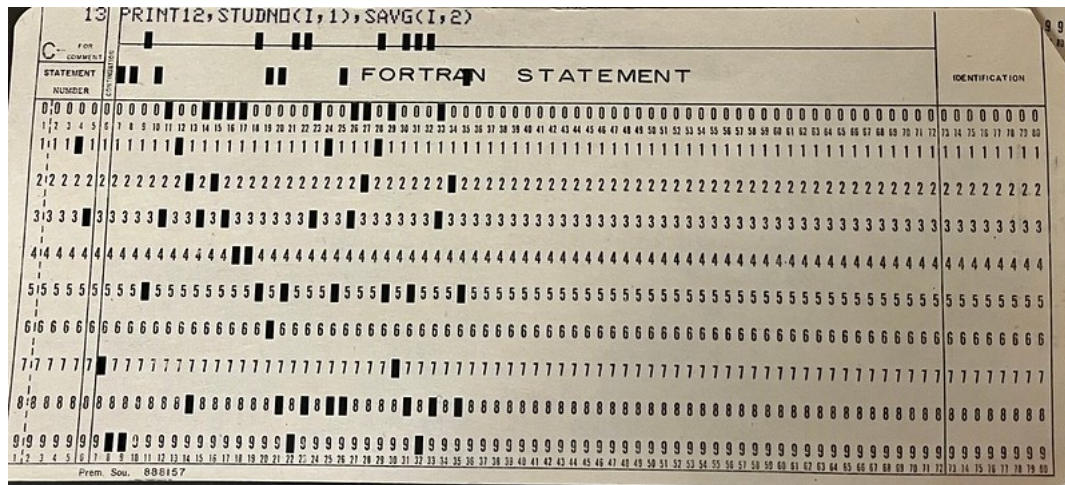


Figure 4: Punched card with FORTRAN statement (at top)

- Understand the relationships of humans and machines
 - Develop critical thinking skills
 - Create games and animations
- The diagram shows different relevant levels of programming and computing including hardware (bottom half) and software (top half). In this course, we're working on "*Applications*" that use the computer to solve problems. Languages other than *Snap!* on this level include *C++*, *Java*, and *Python* (all of these are OOP languages)
 - The top level "Users" refers to most people who only use computers (mostly when operating their smart phones, or driving their cars). As with phones or cars, most of the power is under the hood.
 - I recommend maintaining a (digital) notebook for this course. That's exactly what I did when working through the textbook, using Emacs. (Talk to me if you want to know more about Emacs.)

Programming languages are languages

- C/C++ is like Latin
- SQL is like English

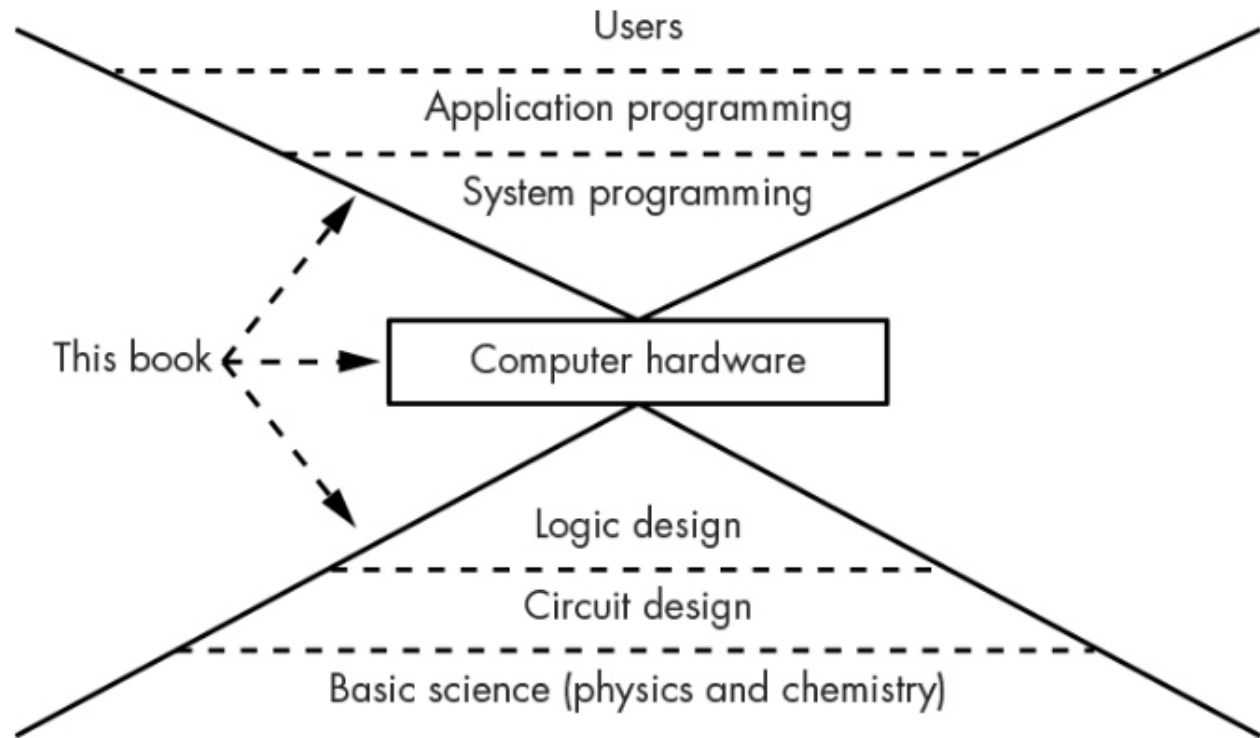


Figure 5: levels of computing (Source: Steinhart, 2019).

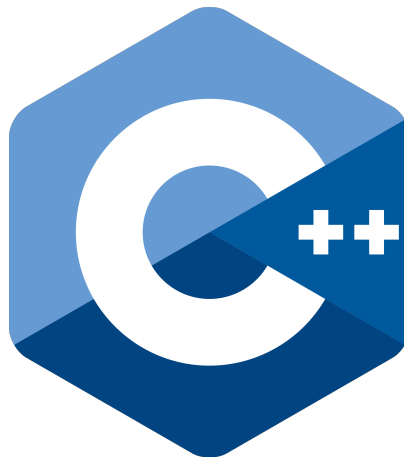


Figure 6: "C/C++ is like Latin"



Figure 7: "SQL is like English"



Figure 8: "Lisp is like French"

- Lisp is like French
- R is like Italian



Figure 9: "R is like Italian"

- Snap! is like Russian or Japanese



Figure 10: "Snap! is like Russian"

- FORTRAN is like Hebrew
- HTML is like Gaelic
- Python is like Spanish

Importance of infrastructure: a look under the hood

Some infrastructure that separates you from just "getting on" with it:

1. Network server / network
2. Keyboard / Screen



Figure 11: "FORTRAN is like Hebrew"



Figure 12: "HTML is like Gaelic"

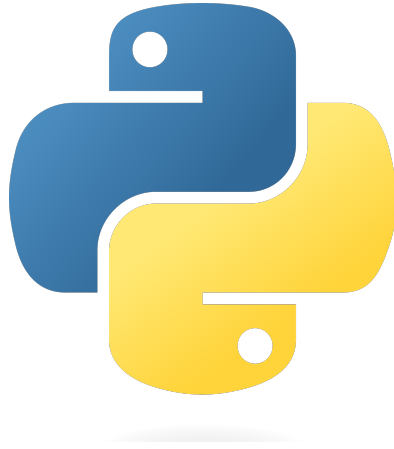


Figure 13: "Python is like Spanish"



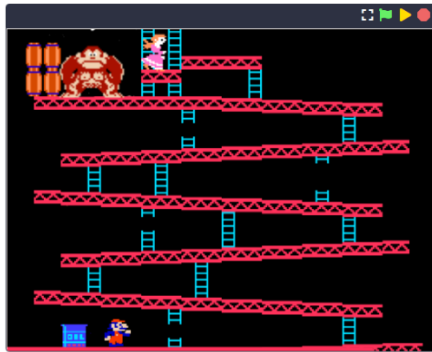
Figure 14: Photo by Landon Martin on Unsplash

3. Operating system (OS)
4. OS shell / terminal / console
5. Python shell / console
6. File system
7. Compiler/interpreter
8. Editor
9. Middleware and meta data
10. Graphical user interface

(Most of these infrastructure components are written in C/C++.)
In Snap!, most of these elements (but not all) are hidden from you.

Why Snap!

Donkey Kong



["Donkey Kong" - Ty Ferguson, Clay](#)

Pacman



["Pacman" - Matthew Wisdom, Tyler Landry, Rylan Turks](#)

Figure 15: Donkey Kong and Pacman arcade games in Snap!

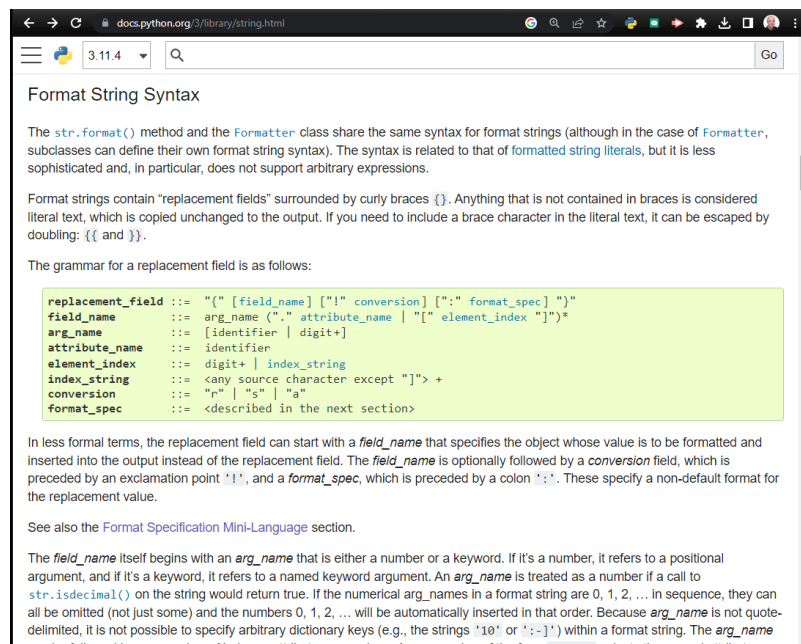
Why Snap!

- It's a full-fledged programming language

- It's easy to build animation and games in it
- It's instantly, freely available online
- It trains pseudocode and modular design
- It's suited for data science applications (multidim arrays)
- It allows you to define recursive functions (see here)
- I've always wanted to get into it

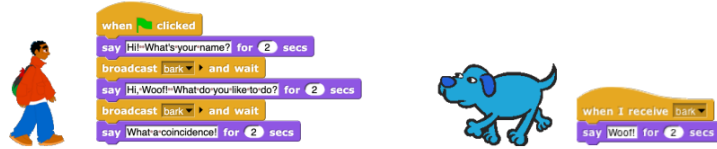
But why not just Python?

- My intuition (better for beginners and tourists)
- Much, much steeper learning curve to get to game design
- Snotty developer community (arrogant nerds aren't nice)
- Compare Python reference vs. Snap! reference manual:



Inter-Sprite Communication with Broadcast

Earlier we saw an example of two sprites moving at the same time. In a more interesting program, though, the sprites on stage will *interact* to tell a story, play a game, etc. Often one sprite will have to tell another sprite to run a script. Here's a simple example:



In the **broadcast bark and wait** block, the word “bark” is just an arbitrary name I made up. When you click on the downward arrowhead in that input slot, one of the choices (the only choice, the first time) is “new,” which then prompts you to enter a name for the new broadcast. When this block is run, the chosen message is sent to *every* sprite, which is why the block is called “broadcast.” (But if you click the right arrow after the message name, the block becomes **broadcast bark to dog and wait**, and you can change it to **broadcast bark to dog and wait** to send the message just to one sprite.) In this program, though, only one sprite has a script to run when that broadcast is sent, namely the dog. Because the boy’s script uses **broadcast and wait** rather than just **broadcast**, the boy doesn’t go on to his next **say** block until the dog’s script finishes. That’s why the two sprites take turns talking, instead of both talking at once. In Chapter VII, “Object-Oriented Programming with Sprites,” you’ll see a more flexible way to send a message to a specific sprite using the **tell** and **ask** blocks.

Notice, by the way, that the **say** block’s first input slot is rectangular rather than oval. This means the input can be any text string, not only a number. In text input slots, a space character is shown as a brown dot, so that you can count the number of spaces between words, and in particular you can tell the difference between an empty slot and one containing spaces. The brown dots are *not* shown on the stage if the text is displayed.

Next: looping, broadcasting, animation

- ☐ Snap user interface (UI)
- ☐ Paint editor
- ☐ Sequence of commands
- ☐ Motion commands
- ☐ Simple looping (repeat, forever)
- ☐ Absolute motion
- ☐ Relative motion
- ☐ Smooth motion using repeat
- ☐ Nested looping
- ☐ XY geometry
- ☐ Costume-based animation

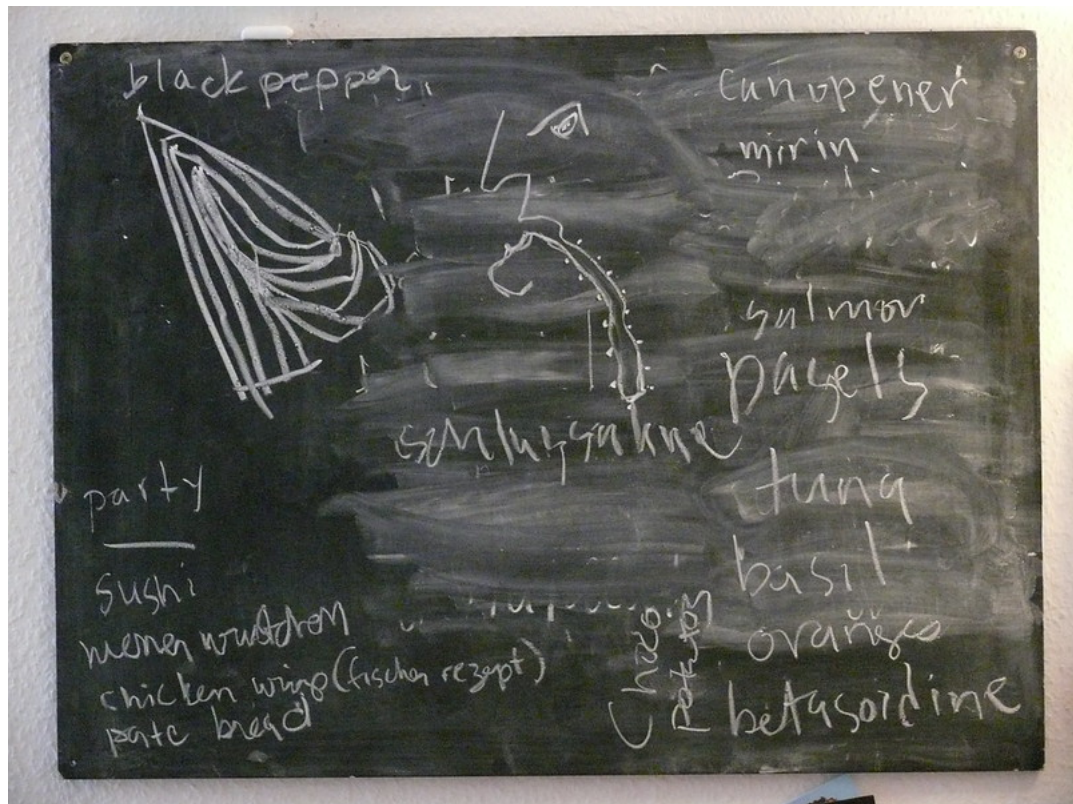
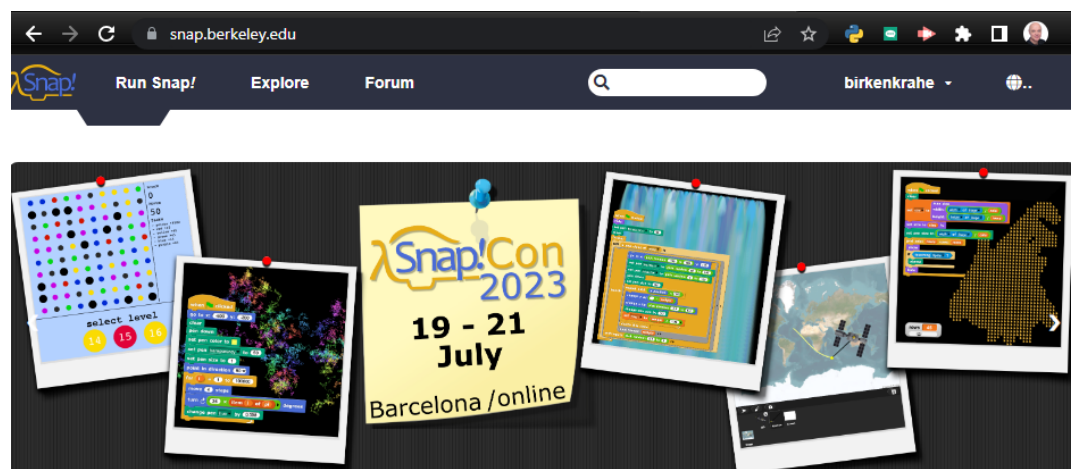


Figure 16: Blackboard user interface (our Berlin kitchen)

What is a User Interface?

- A *user interface* (UI) is the dashboard or platform that allows a user to interact with an application. It's the first thing that you, as a user, see.
- UI/UX is an important, relatively new, interdisciplinary field that includes art and design, usability analysis, etc. UX focuses on the user's path to solving a problem (like shopping online), while UI focuses on the look of the surface of an interactive product (like a web site for online shopping). More: freecodecamp.org video course.

Snap! user interface



Welcome, birkenkrahe!

Snap! is a broadly inviting programming language for kids and adults that's also a platform for serious study of computer science.

[Run Snap! Now](#) [My Projects](#) [My Public Page](#) [Example Projects](#) [Reference Manual](#)

- Connect to *snap.berkeley.edu* and register using your name and Lyon student email address.
- For offline use - on any computer that you can administer, i.e. where you can download and install programs as you please - download the

source files from GitHub, unpack the files, and open `snap.html` in a browser.

- This is how the interface looks like:

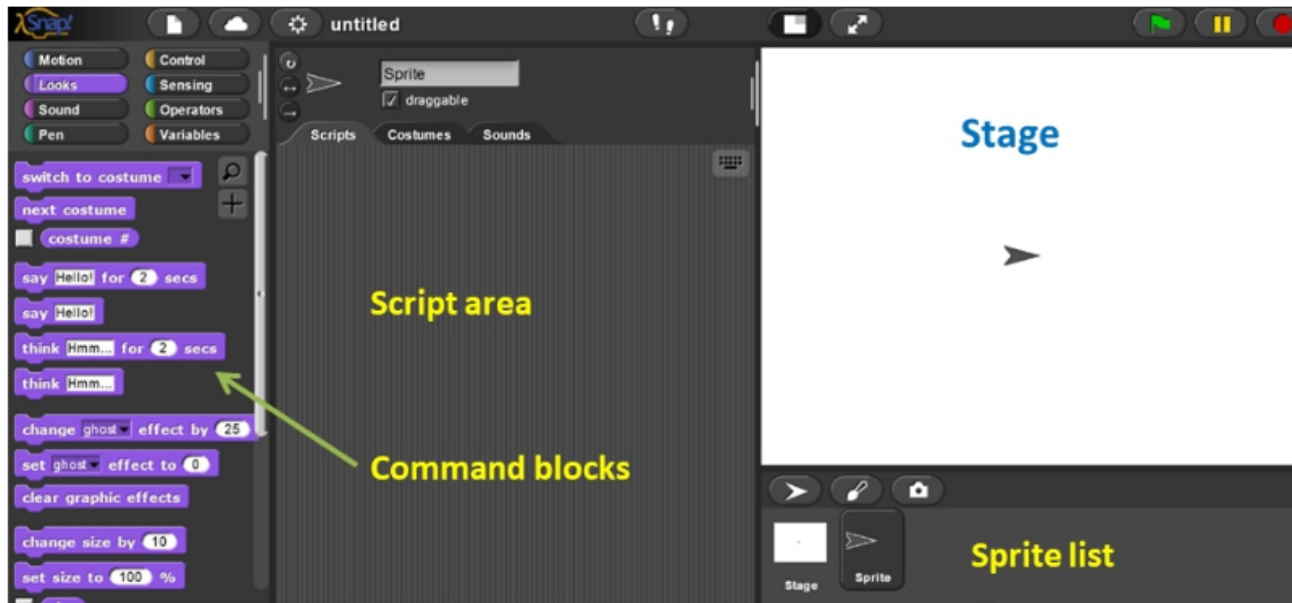


Figure 17: Snap! user interface (Source: Joshi, 2018)

- The interface is reminiscent of a movie maker's studio: *commands* are assembled in the *script* area, and the resulting action plays out on a *stage* with a cast of characters called *sprites*. Every sprite has a script associated with it.
- Compare with Windows Movie Maker - commands on the left, script in the lower half of the screen, sprites/characters in the middle, and stage on the right hand side.
- As a programmer, you are writing the script for each sprite, including movements, sounds, and costumes, but you are also the producer, casting director, and editor.

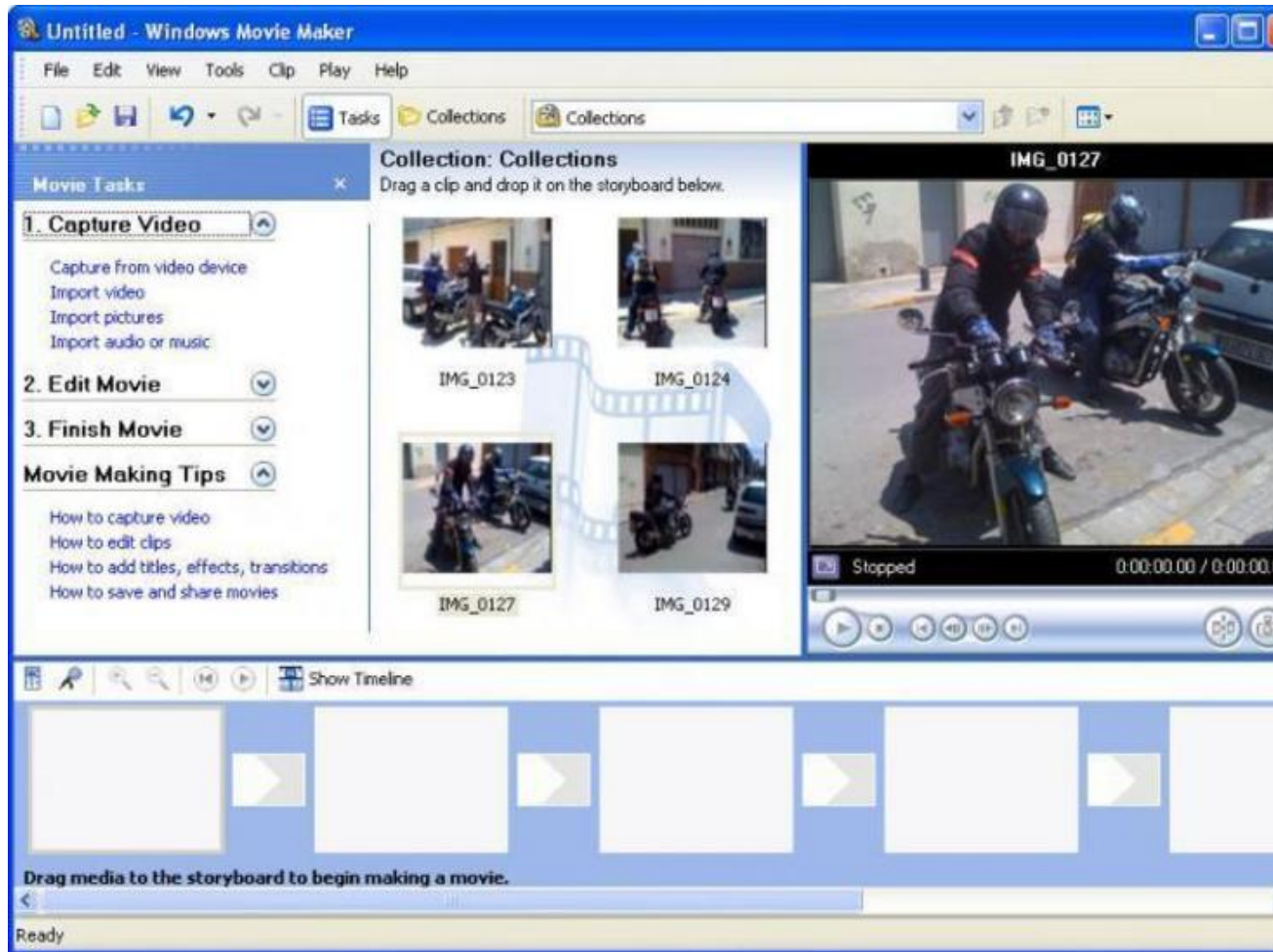


Figure 18: Windows Movie Maker

Summary

- Berkeley's Snap! is a development of MIT's Scratch, created in HTML5 (with JavaScript), available online or on your PC for download.
- In Snap!, you can define multidimensional arrays and recursive functions, which means that you can do anything a high-level language like Python can do, too.
- Programming can help you understand machines, your own thinking, and you can build applications for humans (like games and animations).
- Programming languages are like natural languages, only much stricter.
- Many layers of computing infrastructure separate you from just "getting on with it" - in Snap! you won't have to know most of them.
- A user interface allows a user to interact with an application. UI/UX design is an important, growing career field.