
TIDE 1022

Computational Thinking for Work and Play

Carola Wenk

Department of Computer Science



Jaelle Scheuerman

Newcomb College Institute,
Technology Services



Carola Wenk, Computer Science;
cwenk@tulane.edu

Computational Thinking

- Solve abstract problems using computational approaches
 - Write a computer program
 - Model the problem
 - Develop a sequence of instructions
⇒ Algorithm

Mohammed ibn-Musa *al-Khwarizmi*

In the 9th century AD, he developed “algorithms” for solving linear and quadratic equations. The word “algorithm” stems from the Latin translation of his name.



Algorithms

Give a computational solution to a problem,

- a computer program, or
- a more abstract “algorithm”
(list of instructions)

and reason about its correctness

- Prove that the algorithm solves
the problem.

and efficiency.

- Prove a certain runtime requirement.
- Prove a certain space requirement.

DUTCH APPLE PIE

9-inch unbaked pie shell	Filling:
	2 lb tart cooking apples
	1 tablespoon lemon juice
Topping:	2 tablespoons flour
$\frac{3}{4}$ cup sifted all-purpose flour	$\frac{3}{4}$ cup granulated sugar
$\frac{1}{2}$ cup light-brown sugar, firmly packed	Dash salt
$\frac{1}{2}$ cup butter or margarine	1 teaspoon cinnamon

MAKES 6 TO 8 SERVINGS

1. Prepare pie shell; refrigerate until used.
2. Make Topping: Combine flour and sugar in medium bowl. Cut in butter, with pastry blender or 2 knives, until mixture is consistency of coarse cornmeal. Refrigerate.
3. Preheat oven to 400F.
4. Make Filling: Core apples, and pare; thinly slice into large bowl. Sprinkle with lemon juice.
5. Combine flour, sugar, salt, and cinnamon, mixing well. Toss lightly with apples.
6. Turn filling into unbaked pie shell, spreading evenly. Cover with topping; bake 40 to 45 minutes, or until apples are tender.



VS.



Algorithms

Example problem: Sort ~~n cards~~ numbers in increasing order.

One straight-forward algorithm: Insertion-sort (card-player's sort)

- Incrementally process the numbers,
- maintain a sorted list of numbers seen so far, and
- insert the next number into the sorted list.



Runtime: This generally takes roughly n^2 steps.

More efficient algorithms only take roughly $n \log n$ steps.

Very large data sets and complex computing environments require more sophisticated approaches.

Scratch!

Instead of

```
#include <stdio.h>

int main(){
    printf("Hello world\n");
}
```

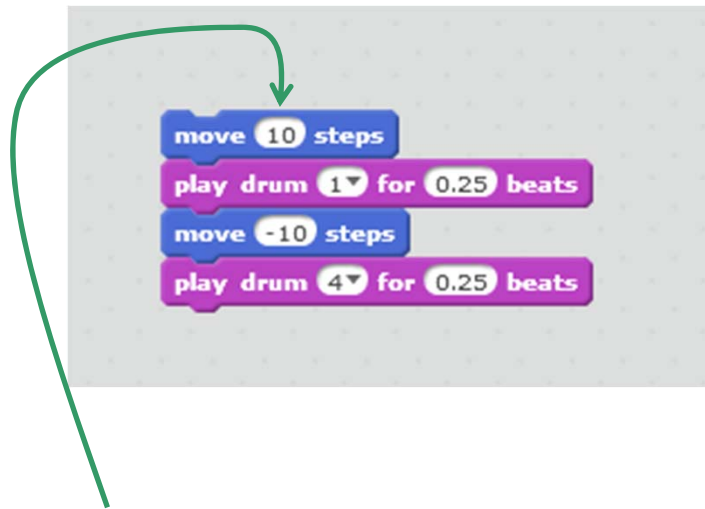
Write programs like this



Go to: scratch.mit.edu

Sequential Execution

Execute multiple instructions **sequentially**; one after the other:



Instructions have **parameters** that can be changed.

So, each of these scratch instructions is in fact a **function**:

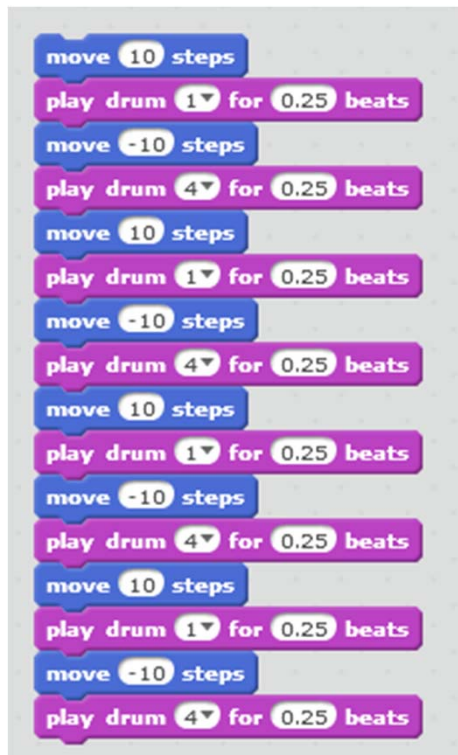
```
move(10)
```

```
play_drum(1, 0.25)
```

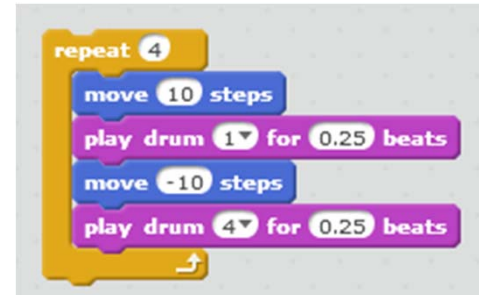
Loops

One can repeat a sequence of instructions ...

... by copy and paste:



...or by using a loop:



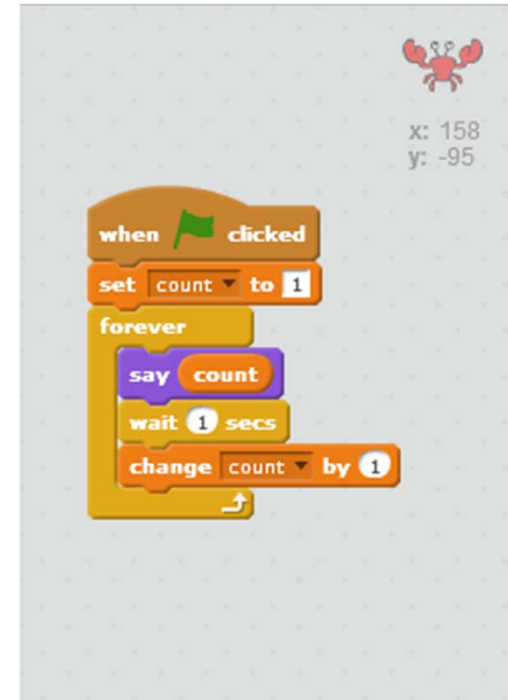
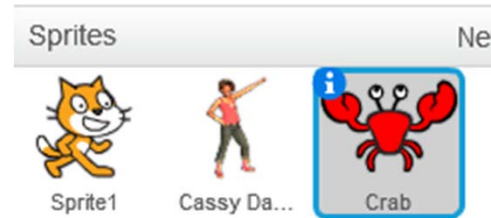
Events

Run a script (or a sequence of instructions) when an **event** happened



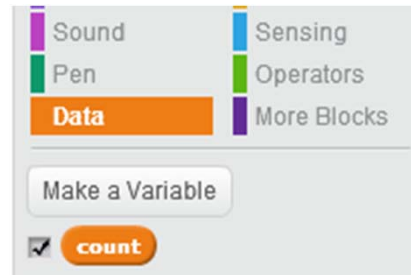
Objects (Sprites)

Each sprite has a separate script.



Variables

Make a variable that stores data:



Store a value in the variable:



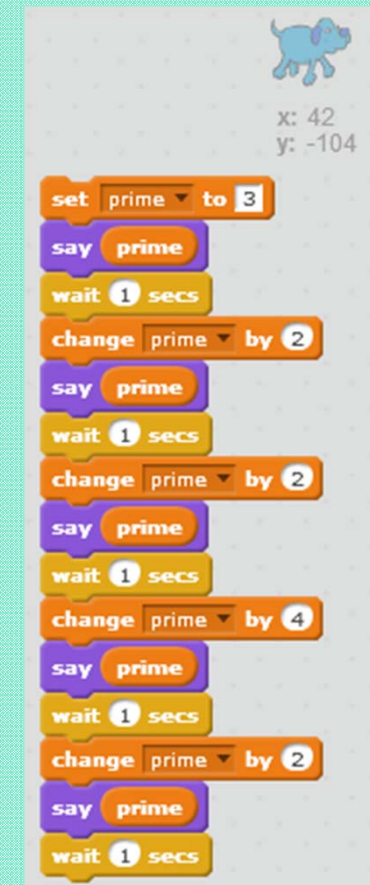
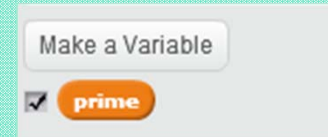
Use the variable:



Modify the variable:

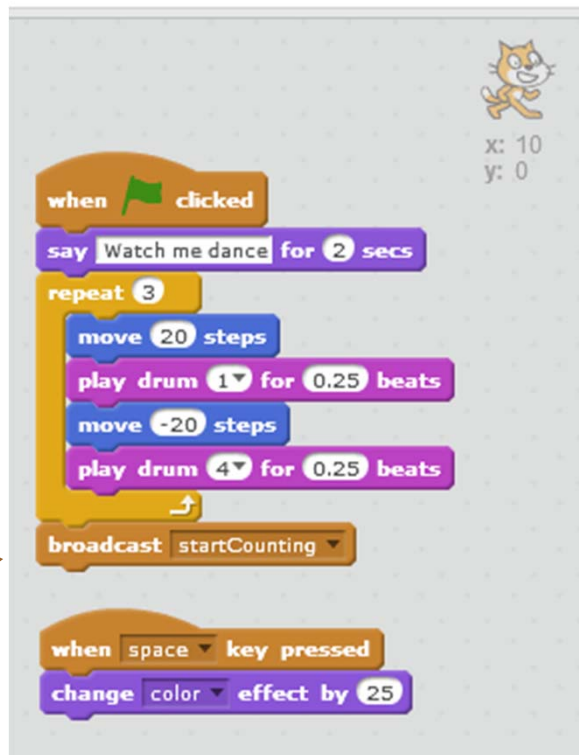


What does this do?



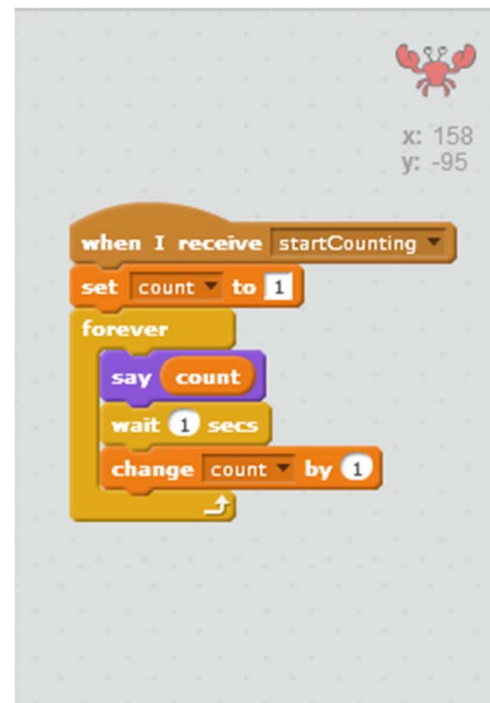
Messages

Coordinate between sprites by sending and receiving **messages**:



Scratch code for a cat sprite (x: 10, y: 0):

- when green flag clicked
- say Watch me dance for 2 secs
- repeat 3
 - move 20 steps
 - play drum 1 for 0.25 beats
 - move -20 steps
 - play drum 4 for 0.25 beats
- broadcast startCounting
- when space key pressed
- change color effect by 25



Scratch code for a crab sprite (x: 158, y: -95):

- when I receive startCounting
- set count to 1
- forever
 - say count
 - wait 1 secs
 - change count by 1