CSC 221: Introduction to Programming

Fall 2018

See online syllabus (also accessible via BlueLine):

http://dave-reed.com/csc221

Course goals:

- To develop problem solving and programming skills to enable the student to design solutions to non-trivial problems and implement those solutions in Python.
- To master the fundamental programming constructs of Python, including variables, expressions, functions, control structures, and arrays.
- To build a foundation for more advanced programming techniques, including object-oriented design and the use of standard data structures (as taught in CSC 222).

What is programming?

programming is *applied problem-solving*

1. understand a problem
2. identify relevant characteristics
3. design an algorithm (step-by-step sequence of instructions to carry out a task)
4. implement the algorithm as a computer program
5. test the program by repeated (and carefully planned) executions
6. GO BACK AND REPEAT/REVISE AS NECESSARY

in short: *programming* is the process of designing, writing, testing and debugging algorithms that can be carried out by a computer

we encounter algorithms everyday: directions to dorm, instruction manual, recipe

- people are smart, so spoken languages can be vague
- computers are not smart, so programming languages are extremely picky

Problem-solving example

Sudoku is a popular puzzle

given a partially-filled 9x9 grid, place numbers in the grid so that

- each row contains 1..9
- each column contains 1..9
- each 3x3 sub-square contains 1..9

		1			2			
	3	7	8					2
2	4						7	3
4						7	1	
			6		8			
	8	2						9
9	5						3	7
6					5	4	8	
			4			5		

how do people solve Sudoku puzzles?

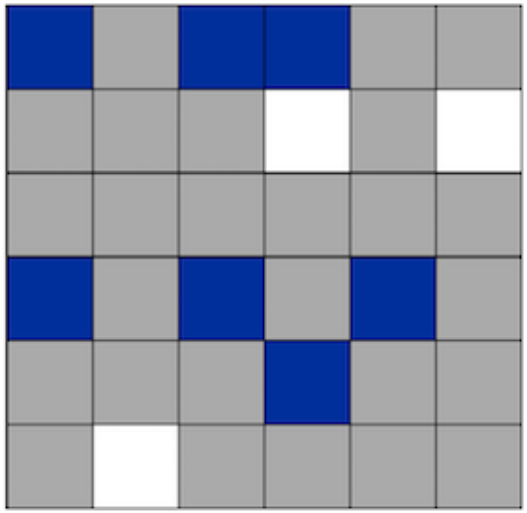
if we wanted to write a program to solve Sudoku puzzles, must/should it use the same strategies?

Related example

3-in-a-row is another puzzle

given a partially-filled NxN grid, color the squares blue or white so that

- each row/column has the same number of each color
- never 3-in-a-row of same color



how similar/different is the human strategy compared to Sudoku?

how similar/different would a computer strategy need to be?

Programming is a means to an end

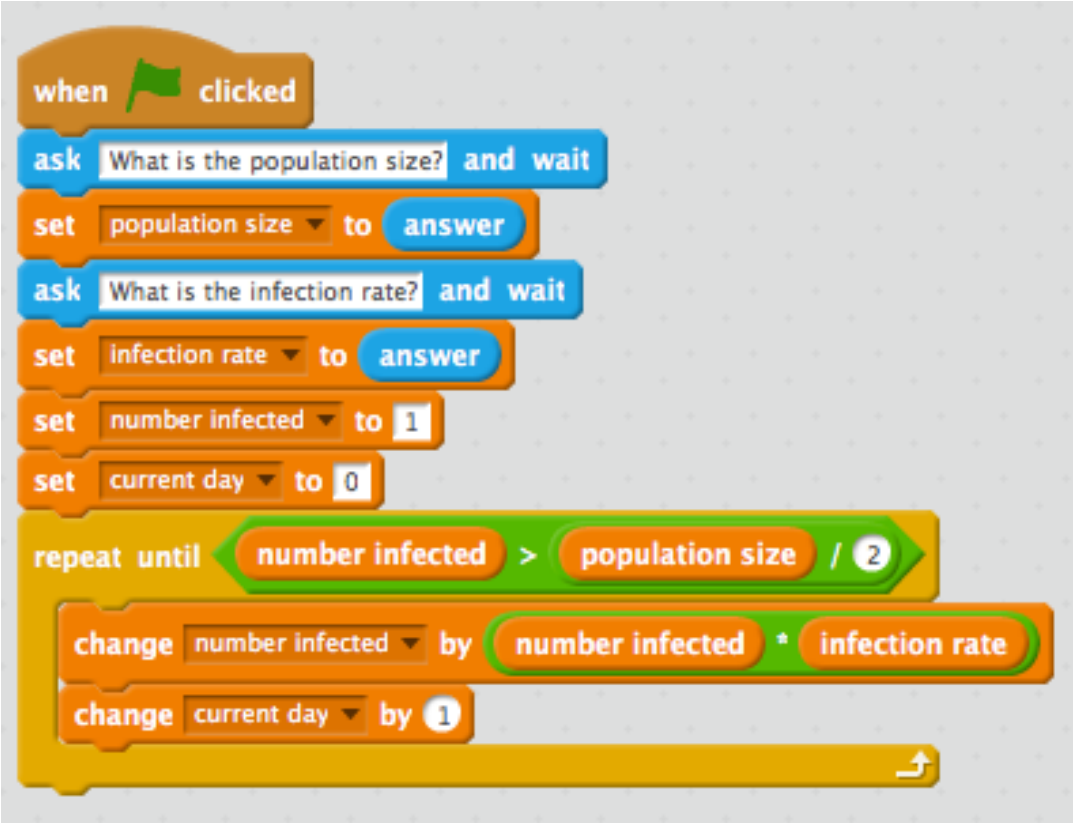
important point: programming is a tool for solving problems

- computers allow people in many disciplines to solve problems they couldn't solve without them
  — natural sciences, mathematics, medicine, business, …
- to model this, many exercises will involve writing a program, then using it to collect data & analyze results

DISEASE VECTOR MODELING: suppose you started with a single infected individual in a population – given a constant infection rate, how long until half the population is infected?

- what information do you need (e.g., population, infection rate)?
- what data values do you need to store and update?
- what is the basic algorithm?
- what real-life factors are we ignoring?

Disease vector model in Scratch



Disease vector model in Python

disease.py - /Users/davereed/Desktop/disease.py

```
population_size = int(input('Population size? '))
infection_rate = int(input('Infection rate? '))
number_infected = 1
current_day = 0

while number_infected <= (population_size/2):
    number_infected += number_infected*infection_rate
    current_day += 1

print('Half of the population is infected by day', current_day)
```

Ln: 13 Col: 0

Another interesting problem

PAPER FOLDING: if you started with a regular sheet of paper and repeatedly fold it in half, how many folds would it take for the thickness of the paper to reach the sun*?

*assuming you had a big enough sheet of paper, a space ship with a forklift, other physically improbable/impossible things…

- what information do you need (e.g., distance of sun)?
- what data values do you need to store and update?
- what is the basic algorithm?

Where do we start?

explore programming concepts using Scratch

- a fun, interactive, online environment for creating animations & games
- we will explore your creative side, while building the foundation for programming
- learn-by-doing, so be prepared to design & experiment & create
- no previous programming experience is assumed
- freely accessible at scratch.mit.edu

SCRATCH

imagine • program • share

will then segue into Python (v. 3) programming

- transfer Scratch programming concepts into a powerful & flexible scripting language
- classes will mix lecture and hands-on experimentation, so be prepared to do things!
- freely downloadable from python.org

python™