Problem-Solving and Abstraction

Fall 2022
Hello, computer
Hello, computer
We use computers every day as electronic *black boxes* that do amazing things by

collecting,

storing,

retrieving, and

transforming *data*. 
Computers only do very basic things.

Numerical calculations:
  Add
  Subtract
  ...

Symbolic manipulations
  Compare two numbers
  Substitute one string of letters and numbers for another
  ...

But when trillions of these simple operations are arranged in the right order, amazing computations can be carried out:

- forecasting tomorrow’s weather
- deciding where to drill for oil
- finding which physical places are most likely to be visited by a person
- figuring out which two people would make a great couple 😍😘

...
A long time ago in a galaxy far, far away....
Computer

From Wikipedia, the free encyclopedia

For other uses, see Computer (disambiguation).

A computer is a digital electronic machine that can be programmed to carry out sequences of arithmetic or logical operations (computation) automatically. Modern computers can perform generic sets of operations known as programs. These programs enable computers to perform a wide range of tasks. A computer system is a "complete" computer that includes the hardware, operating system (main software), and peripheral equipment needed and used for "full" operation. This term may also refer to a group of computers that are linked and function together, such as a computer network or computer cluster.

A broad range of industrial and consumer products use computers as control systems. Simple special-purpose devices like microwave ovens and remote controls are included, as are factory devices like industrial robots and computer-aided design, as well as general-purpose devices like personal computers and mobile devices like smartphones. Computers power the Internet, which links billions of other computers and users.

Early computers were meant to be used only for calculations. Simple manual instruments like the abacus have aided people in doing calculations since ancient times. Early in the Industrial Revolution, some mechanical devices were built to automate long tedious tasks, such as guiding patterns for looms. More sophisticated electrical machines did specialized analog calculations in the early 20th century. The first digital electronic calculating machines were developed during World War II. The first semiconductor transistors in the late 1940s were followed by the silicon-based MOSFET (MOS transistor) and monolithic integrated circuit (IC) chip technologies in the late 1950s, leading to the microprocessor and the microcomputer revolution in the 1980s. The rapid expansion of computing capability has been fueled by increases in computer clock speeds, computer memory, and the size and speed of data communication networks. Computers and computing devices from different eras—clockwise from top left:

- Early vacuum tube computer (ENAAC)
- Mainframe computer (IBM System 360)
- Desktop computer (IBM ThinkCentre S50 with monitor)
- Supercomputer (IBM Summit)
The magic of a computer is its ability to become almost anything you can imagine…

…as long as you can explain *exactly* what that is.
When we program a computer to do something, everything needs to be described precisely.

Say you tell an accounting program to bill your clients the amount each owes.

Should the computer send a weekly bill for $0 to clients who owe nothing?

If you tell the computer to send a threatening letter to clients who haven’t paid, then clients who owe nothing will receive threatening letters until they send you a payment of $0!
When computers behave intelligently, it’s because a person used *their* intelligence to design an intelligent program.
There are many programming languages we can use to write these programs.
github.com/stereoboooster/programming-languages-genealogical-tree
There are many programming languages due to

intended use
history
habit
taste
Ancient history (my childhood)
In this course, we’ll be working in two programming languages:
Why join the navy if you can be a pirate?

famous Steve Jobs quote
— with apologies to Grace Hopper, who was actually a rear admiral in the Navy!
code.pyret.org/editor
1 2
use context essentials2021

```plaintext
include image

circle(30, "solid", "red")
```

...
use context essentials2021

include image

circle(30, "solid", "red")

circle(30, "solid", "yellow")
include image

...circle(30, "solid", "red")

...circle(30, "solid", "yellow")

...above(circle(30, "solid", "red"), circle(30, "solid", "yellow"))

...
use context essentials2021

circle(30, "solid", "yellow")

above(circle(30, "solid", "red"), circle(30, "solid", "yellow"))

above(above(circle(30, "solid", "red"), circle(30, "solid", "yellow")))
Drawing pictures this way is fun – but also a lot of typing.

Here’s where things gets interesting:

*We can define new words.*
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>use context essentials2021</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>g = circle(30, &quot;solid&quot;, 'green')</td>
</tr>
<tr>
<td>4</td>
<td>y = circle(30, &quot;solid&quot;, 'yellow')</td>
</tr>
<tr>
<td>5</td>
<td>r = circle(30, &quot;solid&quot;, 'red')</td>
</tr>
</tbody>
</table>

... above(above(g, y), r)
```kotlin
use context essentials2021

fun traffic-light()
    g = circle(30, "solid", "green")
    y = circle(30, "solid", "yellow")
    r = circle(30, "solid", "red")
    above(y, g, r)
end

traffic-light()
```
In extending the language, the programmer uses the power of functional abstraction to *create new building blocks*. 
fun traffic-light();
g = circle(30, "solid", "green")
y = circle(30, "solid", "yellow")
r = circle(30, "solid", "red")
above(above(g, y), r)
end

fun intersection();
beside(traffic-light(), traffic-light())
end
```kotlin
use context essentials2021

fun traffic-light(size):
    g = circle(size, "solid", "green")
    y = circle(size, "solid", "yellow")
    r = circle(size, "solid", "red")
    above(g, y, r)
end

fun intersection():
    beside(traffic-light(), traffic-light()) beside(traffic-light(), traffic-light())
end
```
use context essentials2021

fun traffic-light(red):
  g = circle(size, "solid", "green")
  y = circle(size, "solid", "yellow")
  r = circle(size, "solid", "red")
  above(above(above(g, y), r)
end

fun intersection():
  beside(traffic-light(), traffic-light())
end

This application expression errored:

8 arguments were passed to the operator.
The operator evaluated to a function defined to accept 5 parameters.

An application expression expects the number of parameters and arguments to be the same.
(Show program evaluation trace...)

Programming as a guest.
use context essentials2021

fun traffic-light(size):
  g = circle(size, "solid", "green")
  y = circle(size, "solid", "yellow")
  r = circle(size, "solid", "red")
  above(above(g, y), r)
end

fun intersection(size):
  beside(traffic-light(size), traffic-light(size))
end
What will we do in this course?
1. Identify and organize the data needed to solve a problem
   - Data design

2. Break a problem down into subproblems that can be solved with computations
   - Programming

3. Express computations over the data
   - Programming/CS

4. Test those computations to make sure they’re doing what they’re supposed to
   - Testing
Think about whether it’s a good idea to solve the problem, and how your solution might affect the world around you.
This course teaches skills that will help you both in computer science and beyond.
Three major units:

Tabular / data science data
Other core CS data structures and how to program with them
Additional data structures and programming techniques

Pyret  Python
Goals

Apply fundamental data-organizations (called data structures) to capture the information in a computing problem.

Break down a computing question into manageable smaller problems.

Write programs to compute answers to questions over fundamental data structures.

Check whether your programs behave as intended/requisite.
Course information
Class:
  Monday & Wednesday, 9:00–10:15 a.m.
  Sanders Classroom 006

Lab:
  Friday, 9:00–11:00 a.m.
  Sanders Classroom 006
CS1: Problem-Solving and Abstraction

CS101 Section 02
Vassar College, Fall 2022
Syllabus / Course Wiki

Welcome to our course and syllabus. It will be updated throughout the semester with important course information, so check here regularly.

Contact Information

Professor: Marc Smith
Office: SP 104.5
Virtual: 🎤 Zoom
Phone: 845.437.7497
Email: mlsmith (best way to contact me)

Course Coordinates

Lectures: Mon/Wed 9:00-10:15am in SC 106
Labs: Fri 9:00-11:15am in SC 106
Office Hours: by appointment
Wiki: https://www.cs.vassar.edu/courses/cs101-2022-02/top
Gradecorner: https://www.gradecorner.com/
CMPU 101 §02
Computer Science I:
Problem-Solving and Abstraction

Fall 2022

Monday  9:00–10:15 a.m.
Wednesday 9:00–10:15 a.m.
Friday  9:00–11:00 a.m.
Sanders Classroom 006

Professor Smith

cs.vassar.edu/courses/cs101-2022-02/top

Overview

This course introduces fundamental concepts of computer science. Its major goal is to introduce students to the principles of systematic problem-solving through programming. We will explore the art and science of problem-solving using the computer as a tool, writing programs in Pyret – a simple yet powerful student learning language –
Grading

Weekly lab exercises 20%
Grading

Weekly programming assignments
CMPU-101-03
Computer Science I: Problem-Solving and Abstraction

DESCRIPTION
Edit your course description on the Course Settings page.

THINGS TO DO
1. Add students or staff to your course from the Roster page.
2. Create your first assignment from the Assignments page.

ACTIVE ASSIGNMENTS | RELEASED | DUE (EDT) | SUBMISSIONS | % GRADED | PUBLISHED | REGRADES

You currently have no assignments.
Create an assignment to get started.

Create Assignment
Vassar CS Student Integrity Guide

This guide is designed to clarify Vassar College’s academic integrity policy as it applies to the Computer Science Department. Furthermore, it provides advice on how to best navigate integrity issues in the context of the field, where source code authorship is a central issue.

The goal of our computer science courses is to promote understanding of the field, not competition among students. As such, students are encouraged to discuss class material, ideas, sample exercises, etc., with other students.

However, when it comes to graded work (e.g., programming assignments, programming labs, take-home exams), it is important to know when to collaborate and when to work individually. Taking shortcuts, while seemingly beneficial in the short term, will inevitably backfire later on. Conversely, the challenges of working through a problem will pay off greatly in future courses and postgraduate life, as they will enable students to be more independent in their work.

1. Policy

1.1. Guidelines for individual work

The goal of individual work is to assess the learning of each person in isolation. The guidelines are the following:
Grading

- Labs: 20%
- Assignments: 20%
Grading

Two midterm exams
Grading

- Labs: 20%
- Assignments: 20%
- Exam 1: 20%
- Exam 2: 20%
- Exam 3: 20%

Regularly scheduled final exam
“All through our education, we are being taught a kind of reverse mindfulness. A kind of Future Studies where – via the guise of mathematics, or literature, or history, or computer programming, or French – we are being taught to think of a time different to the time we are in. Exam time. Job time. When-we-are-grown-up time.

To see the act of learning as something not for its own sake but because of what it will get you reduces the wonder of humanity. We are thinking, feeling, art-making, knowledge-hungry, marvelous animals, who understand ourselves and our world through the act of learning. It is an end in itself. It has far more to offer than the things it lets us write on application forms. It is a way to love living right now.”

Matt Haig, *Notes on a Nervous Planet*
So...

Don’t just focus on grades.

Don’t just focus on what happens if you get an A in the course.

Be here now.

If you worry about whether you understand what we’re doing in class, in lab, and on the assignments, your grades will take care of themselves.

Trust me.
How do you draw so well?

Practice.

It must be an innate gift...
A gift from God...

It's practice.

I'll never understand how some people are so talented...
A mystery...

Practice.

© Sarah Andersen
We’ve got a big journey ahead of us. I hope you’re excited!
Acknowledgments

This class incorporates material from:

Peter J. Denning and Matti Tedre, *Computational Thinking*

Kathi Fisler and Doug Woos, Brown University

W. Daniel Hillis, *The Pattern on the Stone*