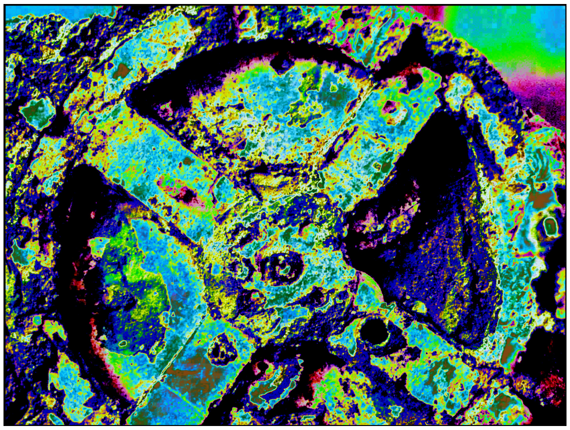


CS1101 Computer Science I

Spring 2016

Robert Muller
Boston College

Computer Science I

Today

- What this course is about
- Logistics
- Course administration

Computer Science I

Super TA Staff **(03 OCaml)**

Nick Denari
Lab 03 Higgins 280
Tuesdays 4PM

Meagan Gonzalez
Lab 01 Higgins 280
Wednesdays 3PM

Computer Science I

Super TA Staff **(04 Python)**

Laura Baumgartner
Lab 04 Higgins 280
Wednesdays 10AM

Jesse Mu
Lab 02 Higgins 280
Tuesdays 5PM

Computer Science I

Home

http://www.cs.bc.edu/~muller/teaching/cs1101/s16/

CS 1101 Computer Science I

Fall 2015

Computer Science Department
The Morrissey College of Arts and Sciences
Boston College

About Staff Textbook Grading Schedule Resources
Notes Labs Piazza Canvas GitHub Problem Sets
Manual StdLib UniLib OCaml.org

Created on 08-31-2015 20:41.

Computer Science I

What CS1101 is About

Three interwoven themes:

1. Learning about information & computation
2. Learning how to **code**
3. An introduction and gateway to *Computer Science*

Learning how to **code**

- Application of logic in problem solving (math-ish)
- *Clear, concise expression of ideas/ algorithms (english/poetry-ish)*

Learning how to **code**

- Have an idea? You can build it!
- Empowering in almost any field ($$)
- Interesting and *really* fun!
- Learn by doing!

Learning how to **code**

- We'll use:

  [~~Basic~~; ~~Pascal~~; ~~C~~; ~~Java~~; ~~Python~~]

  OCaml! as our programming language

Why OCaml?

- Computation can be approached from either a mathematical or mechanical perspective
- From the former perspective, coding is a *natural extension of algebra*

Why OCaml?

- Ocaml emphasizes the most important ideas:
  - expression reduction/simplification,
  - functions, abstraction & composition,
  - variables are mathematical variables,
  - types.

Why OCaml?

- Languages in industry adopting ideas from ML:

  [Java 8; C#; F#; Python; JavaScript; C++; Rust; Go; Elm; Swift; Scala; …]

- Not that it matters, but other good schools doing likewise.

Required Work

- Two 75-minute lectures each week
  open laptops prohibited!
- One 50-minute lab each week
  laptops required!
- Ten programming projects, time requires varies but expect 8-10 hours of work each week,
- Three exams.

Take-Aways

- By the end of the semester:
  - You'll have a reasonably robust understanding of computation
  - You'll be **skilled**; able to think "computationally" able to **code**!
  - You'll have a better understanding of computer science.

Take-Aways

- By the end of the semester:
  - You'll be a competent beginning programmer and will be able to pick up Python or Java easily;
  - You'll be well-prepared for CS1102;
  - You'll have a better understanding of computer science.

Required Background

- High School algebra
- Familiarity with basic trigonometry and geometry also helpful.
- No programming experience required.
- A taste for building things also helpful.

Computation and Calculation

Three Aspects of Computation

1. Simplification
2. Abstraction
3. Composition

Simplification

In middle school we learned about algebraic expressions:

$$ax^2 + bx + c$$

Where a, b and c are **constants** and x is a **variable**. We learned to solve for roots, how to factor them, we learned the properties of their curves, etc.

Simplification

For example, letting the constants a = 3, b = 2 and c = 1, we have:

$$3x^2 + 2x + 1$$

Which has fixed constants and a variable x.

Simplification

We can plug a number in for variable x and simplify. Say 5:

$3 \bullet 5^2 + 2 \bullet 5 + 1$

Simplification

$3 \bullet 5^2 + 2 \bullet 5 + 1$

➔ $3 \bullet 25 + 2 \bullet 5 + 1$

➔ $75 + 2 \bullet 5 + 1$

➔ $75 + 10 + 1$

➔ $85 + 1$

➔ 86

Simplification

$3 \bullet 5^2 + 2 \bullet 5 + 1$

➔ $3 \bullet 25 + 2 \bullet 5 + 1$

➔ $75 + 2 \bullet 5 + 1$

➔ $75 + 10 + 1$

➔ $85 + 1$

➔ 86 ⬅ A value

Simplification

$3 \bullet 5^2 + 2 \bullet 5 + 1$

5 units of work in 5 steps

→ $3 \bullet 25 + 2 \bullet 5 + 1$

→ $75 + 2 \bullet 5 + 1$

→ $75 + 10 + 1$

→ $85 + 1$

→ 86

Computer Science I

Parallel Simplification

$3 \bullet 5^2 + 2 \bullet 5 + 1$

5 units of work In **3** steps

→ $3 \bullet 25 + 10 + 1$

→ $75 + 11$

→ 86

Computer Science I

Abstraction

Algebraic expressions packaged up as *functions*:

$$f(x) = 3x^2 + 2x + 1$$

We can take this as a *definition* of function f.

Computer Science I

Function Definitions and Uses

Euler's notation for **uses**, **calls** or **applications** of function f:

f(5)   f(2 + 2)

1. Simplify the argument to value V,
2. plug the value V in for x,
3. simplify the result.

Computer Science I

Simplification

f(2+2) → f(4)
→ $3 \bullet 4^2 + 2 \bullet 4 + 1$
→ $3 \bullet 16 + 2 \bullet 4 + 1$
→ $48 + 2 \bullet 4 + 1$
→ $48 + 8 + 1$
→ $56 + 1$
→ 57

Computer Science I

Functions and Code

- Roughly speaking, a piece of computer software is a collection of functions.
- In HS algebra our functions usually worked with real numbers.
- In programming, there are lots and lots of interesting **types** of inputs for our functions.

Computer Science I

Code

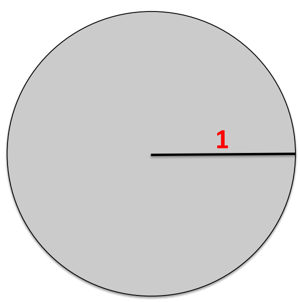
OCaml:

```
let f x = 3 * x ** 2 + b * x + c
```

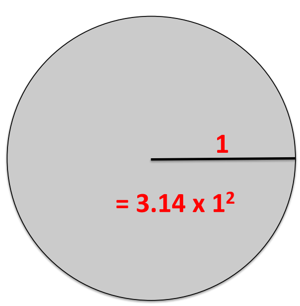
Python:

```
def f(x):
    return 3 * x ** 2 + b * x + c
```

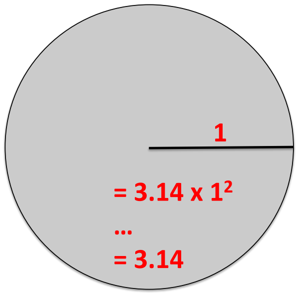

Example: area of unit circle



Example: area of unit circle

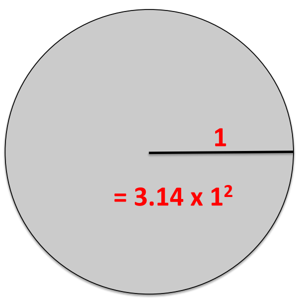


Example: area of unit circle



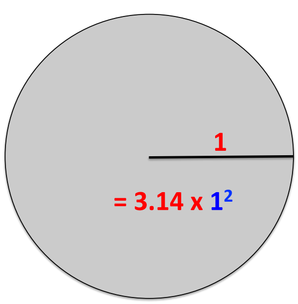
Computer Science I

Example: area of unit circle



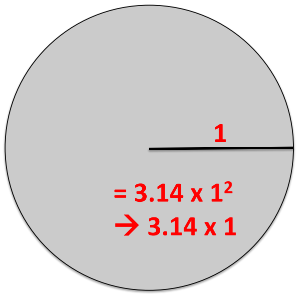
Computer Science I

Example: area of unit circle



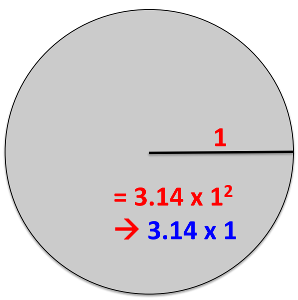
Computer Science I

Example: area of unit circle



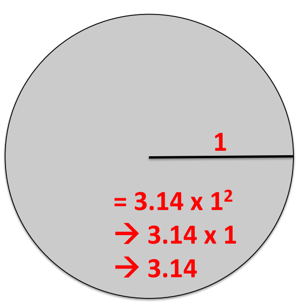
Computer Science I

Example: area of unit circle



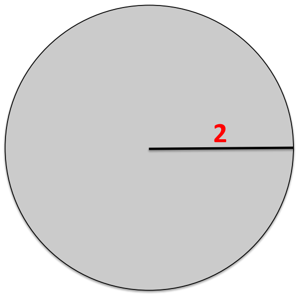
Computer Science I

Example: area of unit circle

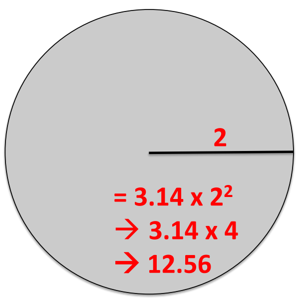


Computer Science I

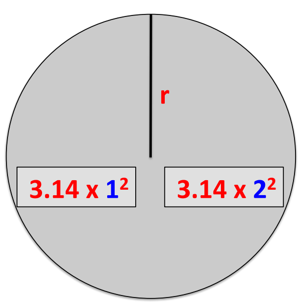
Example: area of circle of radius 2



Example: area of circle of radius 2

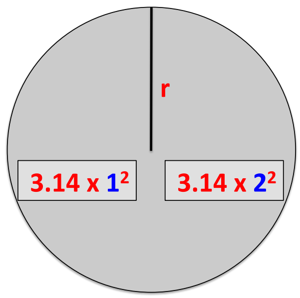


Example: area of circle of radius r



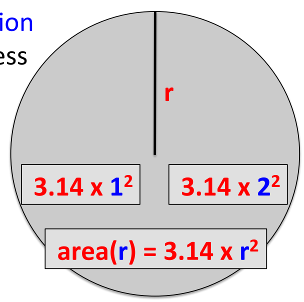
Example: area of circle of radius r

We want to *abstract* with respect to the variation(s).



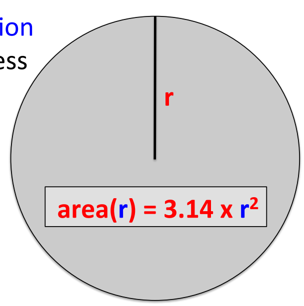
Example: area of circle of radius r

A function definition allows us to express the *abstraction*.



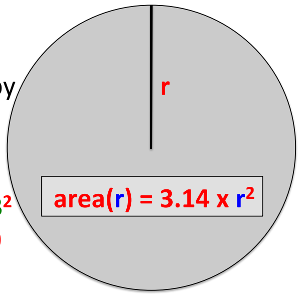
Example: area of circle of radius r

A function definition allows us to express the *abstraction*.



Example: area of circle of radius r

Our function definition can now be *used* or *called* by providing an input.



area(3) → 3.14 x 3^2
→ 3.14 x 9
→ 28.26

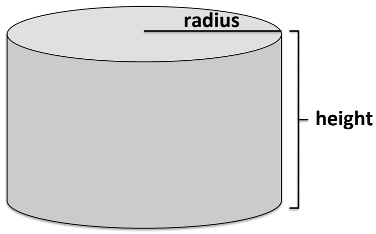
In OCaml

```
# let area radius = 3.14 *. radius ** 2.0;;
val area : float -> float = <fun>

# area(2.0);;
- : float = 12.56
```


In OCaml

```
# let area radius =
    let pi = acos (-1.)
    in
    pi *. radius ** 2.0;;

val area : float -> float = <fun>
```

Example: volume of a cylinder



In OCaml

```
let area radius =
  let pi = acos (-1.)
  in
  pi *. radius ** 2.0

let volume radius height =
   (area radius) *. height
```


Euclid's GCD Algorithm, 300BCE

$$gcd(m, n) = \begin{cases} m & \text{if n is 0,} \\ gcd(n, m \% n) & \text{otherwise} \end{cases}$$

Euclid's GCD Algorithm, 300BCE

$$\gcd(m, n) = \begin{cases} m & \text{if n is 0,} \\ \gcd(n, m \% n) & \text{otherwise} \end{cases}$$

```
gcd(25, 10) = gcd(10, 25 % 10)
            = gcd(10, 5)
            = gcd(5, 10 % 5)
            = gcd(5, 0)
            = 5
```


Euclid's GCD Algorithm, 300BCE

$$\gcd(m, n) = \begin{cases} m & \text{if } n = 0, \\ \gcd(n, m \% n) & \text{otherwise} \end{cases}$$

```
let rec gcd m n =
   match n = 0 with
    | true  -> m
    | false -> gcd n (m mod n)
```

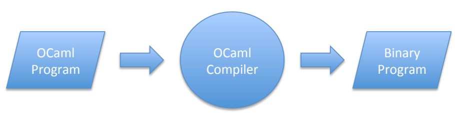

CS101 and CS102

- A principal theme of CS101 is mastering the art of expressing algorithms as functions, **procedural abstraction**.
- A principal theme of CS102 is mastering the art of writing new types, (values and functions), **data abstraction**.

How Programming Works

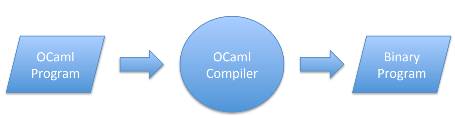
- Using an *editor* program, a programmer develops the TEXT of a program in some language, e.g., OCaml or Python
- They then use another program, a *compiler*, to *translate* the text into the binary language of the machine.

Programming (Basic Model)



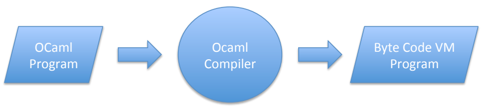
Binary Program is in the native language of the computer so the binary program can be executed.

Programming (Basic Model)



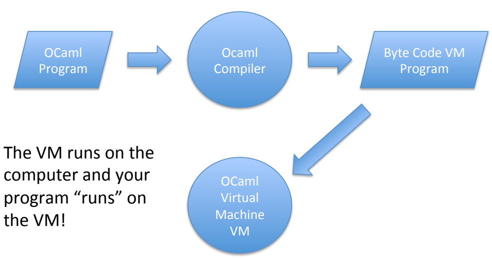
Since each computer has it's own native language, a compiler that can produce binaries for one computer won't necessarily be able to produce binaries that will run on a different computer.

Programming (VM Model)



The Byte Code Program is in the native language of a "virtual" computer. The virtual machine (VM) is just a program that can be implemented on any computer, no matter it's binary language.

Programming (VM Model)



The VM runs on the computer and your program "runs" on the VM!

Course Admin

Course Admin

- Two 75-minute lectures each week;

  No laptops/screens in lecture.

- One one-hour lab each week;

  Laptops required in lab.

NB: FIRST LABS MEET **THIS WEEK**.

Tour of course website

Resources

- Extensive lecture notes
- Most of our material is covered in lecture, background reading in *OCaml from the Beginning*.
- Office hours, Piazza, the internet, your colleagues

Grading

- 46% for 10 problem sets, plenty of opportunity for extra credit
- 42% for 3 exams
- 12% for consistent course participation
  - Lab, lecture, Piazza forum

How to Succeed in CS 1101

- Start problem sets *right away*!
- Pay careful attention to detail.
- Seek help when you need it.
- Show up consistently, participate in class, ask questions.

Rules of the Road

- Late homework penalty 25% each day, penalty excused for documented medical problems or family emergencies only;
- Honor code strictly enforced.