CS1101S Studio Session Week 2: *Computation, Source Language & Abstraction*

Niu Yunpeng

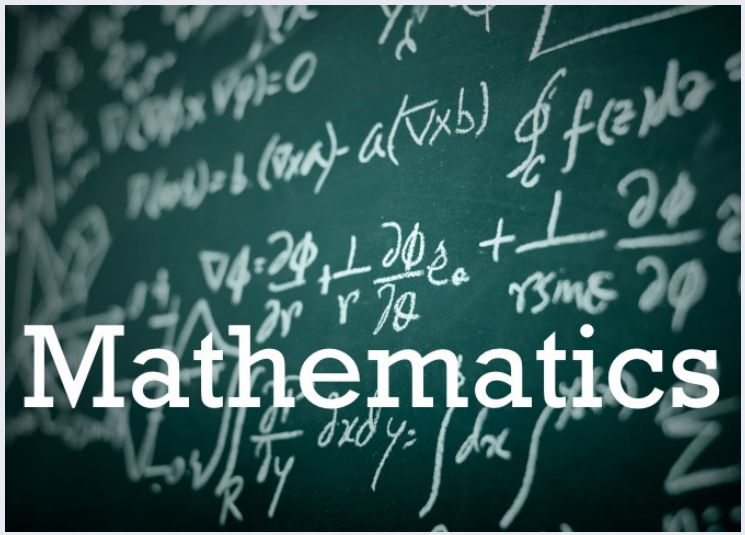
niuyunpeng@u.nus.edu

August 21, 2018

Overview

Computer Science & Computation

Mathematics vs Computer Science



Computer Science & Computation

Mathematics

- The declarative knowledge
- The "what-is" knowledge
- Defines what the problem is

Computer Science

- The imperative knowledge
- The "how-to" knowledge
- Tells how to solve the problem

Square root for a mathematician

The square root of a non-negative number x is a non-negative number y such that the square of y is x. Symbolically, for every non-negative number x, $y = \sqrt{x}$ if $x = y^2 \cap y \geq 0$.
(We do not consider complex numbers here.)

Square root for a computer scientist

In order to find an approximation of $\sqrt{x}$,

- Make a guess of y;
- Calculate the average of y and x/y;
- Keep improving the guess until it is good enough.

Computer Science & Computation

From Computer Science to Computation

- To write a program = to express a **computational process**.
- Usually, we prefer a more effective **computational process**.
- A **computational processes** is composed of many procedures, each of which is a program.

Computation & Programming Language

How to communicate a mathematical process

- After hundreds of years, mathematicians have defined a full set of notations to express the mathematical communication formally.
- The most basic ones are $+, -, \times, \div$.

How to communicate a computational process

- Although Computer Science is much younger, we did/are doing/will continue to do similar things.
- They are called **programming languages**.

Computation & Programming Language

To summarize

- Computation - the process of solving problems
- Program - the individual procedure of the computational processes
- Programming language - the tool to communicate in CS

Overview

The Source Language

About the Source

- Offical tailor-made programming language for CS1101S.
- A sub-language of JavaScript.
- Used to be called JediScript.

Source Playground

- Standalone version at `https://sourceacademy.nus.edu.sg/playground/`.
- Embedded version in the Source Academy.

The Source Language

Components of a programming language

- Primitives
- Combination
- Abstraction

The Source Language

Components of a programming language

- Primitives:
  The smallest constituent unit of a programming language.
- Combination:
  Ways to put primitives together.
- Abstraction:
  The method to simplify the messy combinations.

Details - Primitives

Primitive Data

- Numerals:
- Booleans:
- Strings:

Primitive Procedures

- Basic algebra:

Details - Primitives

Primitive Data

- Numerals: 6, -54, 0, 123.45, 11.5e2, NaN, etc.
- Booleans: true, false
- Strings: "Singapore", "N", '1101', etc.

Primitive Procedures

- Basic algebra: $+, -, \times, \div, \%$.

Details - Primitives

What are primitives?

- The smallest constituent unit of a programming language.

How to understand?

- The story of **atom** in chemistry.
- The story of **primitive** in CS.

Details - Combination

Means of combination

Of course, just put primitives together, "combine"!

Wait, how to put them together?

Apply operators on operands (and thus become an expression).

Details - Combination

A simple example

- Operand: 1, 2
- Operator: $+$
- Expression (result of combination): $1 + 2$

Details - Combination

But, is this enough?

No, operands can become combination as well.

Another example

- Operand: $1 + 2$, $3 + 4$
- Operator: *
- Complex expression (combination of combination): $(1 + 2) * (3 + 4)$

Details - Combination

More operators

- Arithmetic operators:
- Comparison operators:
- Boolean operators:
- Conditional operators:

Details - Combination

More operators

- Arithmetic operators: $+, -, \times, \div, \%$.
- Comparison operators: $>, <, \geq, \leq, ===, !==$.
- Boolean operators: $\&\&, ||, !$.
- Conditional operators: `<stmt-a> ? <stmt-b> : <stmt-c>`.

Caution

What is the difference between $=$, $==$ and $===$?

Details - Combination

Is combination really that simple?

- Maybe yes, if you are adding two integers.
- But, what if you need to add two complex numbers?
- What if you need to add two vectors?
- What if you need to add two electrical signals?
- ...

Details - Combination

To point out

- The same combination can be appiled to
  - Two very simple objects; or
  - Two very complicated objects.
- Combination shall provide a generic interface (a convention) so that
  - We can (theoretically) combine everything as long as the convention is not broken.

Details - Abstraction

Means of abstraction

- To abstract data: use naming;
- To abstract procedures: use functions.
- Sometimes, naming and functions are combined together.

Details - Abstraction

Naming

- To give a name to some data.
- When referring to that data in the future, use its name instead.
- In Source, use `const name = data;` to name a new constant.
  - There are more ways of naming, such as using `let` and `var`.
  - We will touch them in a few weeks' time.

Functions

- To abstract a procedure: use functions.
- Two steps to use a function: define a function, apply a function.

Details - Abstraction

Example

- Given the radius of a circle, please write a function to calculate the area of this circle.

Notice

- You are only allowed to use the 2^{nd} abstraction technique: **functions**.

Details - Abstraction

Answer

- The area of a circle with a radius of 3:

```
( function (x) { return 3.14159 * x * x; } )(3);
```

- The area of a circle with a radius of 5.6:

```
( function (x) { return 3.14159 * x * x; } )(5.6);
```

Details - Abstraction

Naming of functions

- Waste of time to repeat writing the same expressions.
- Solution: Give them names.
- Why: Combination of means of abstraction.

Details - Abstraction

Example again

- To calculate the area of a circle:

```
const pi = 3.1415926535;
const circle = function (x) { return pi * x * x; };
```

Thus..

- The area of a circle with a radius of 3: `circle(3);`
- The area of a circle with a radius of 5.6: `circle(5.6);`

Details - Abstraction

Naming of functions - another way to write

```
const pi = 3.1415926535;

function circle(x) {
    return pi * x * x;
}
```

To use them - the same

- The area of a circle with a radius of 3: `circle(3);`
- The area of a circle with a radius of 5.6: `circle(5.6);`

Details - Abstraction

Naming of functions - use pre-defined constants

```
function circle(x) {
    return math_PI * x * x;
}
```

To use them - the same

- The area of a circle with a radius of 3: `circle(3);`
- The area of a circle with a radius of 5.6: `circle(5.6);`

The Source Language

To summarize

- Primitives: primitive data & primitive procedures;
- Combination: expression = operands + operators;
- Abstraction: naming (for data) & functions (for procedures).

The Source Language

A few terms before we continue...

- Solution to a problem
- Computational Process
- Program
- Statement
- Expression

The Source Language

A few terms before we continue...

To find the **solution to a problem**, we are *essentially* trying to find a **computational process**. This process is usually described by a **program**. A program consists of many **statement**s, each of which ends with a **semicolon**.

The Source Language

Why the semicolon ";"?

- A statement is an instruction to tell the Source to execute something.
- That "something" is a kind of basic action, which is the *expression*.
- Therefore, the semicolon changes an *action* into an *instruction* (so changes an expression into a statement).

Overview

Black-box Abstraction

The black-box concept

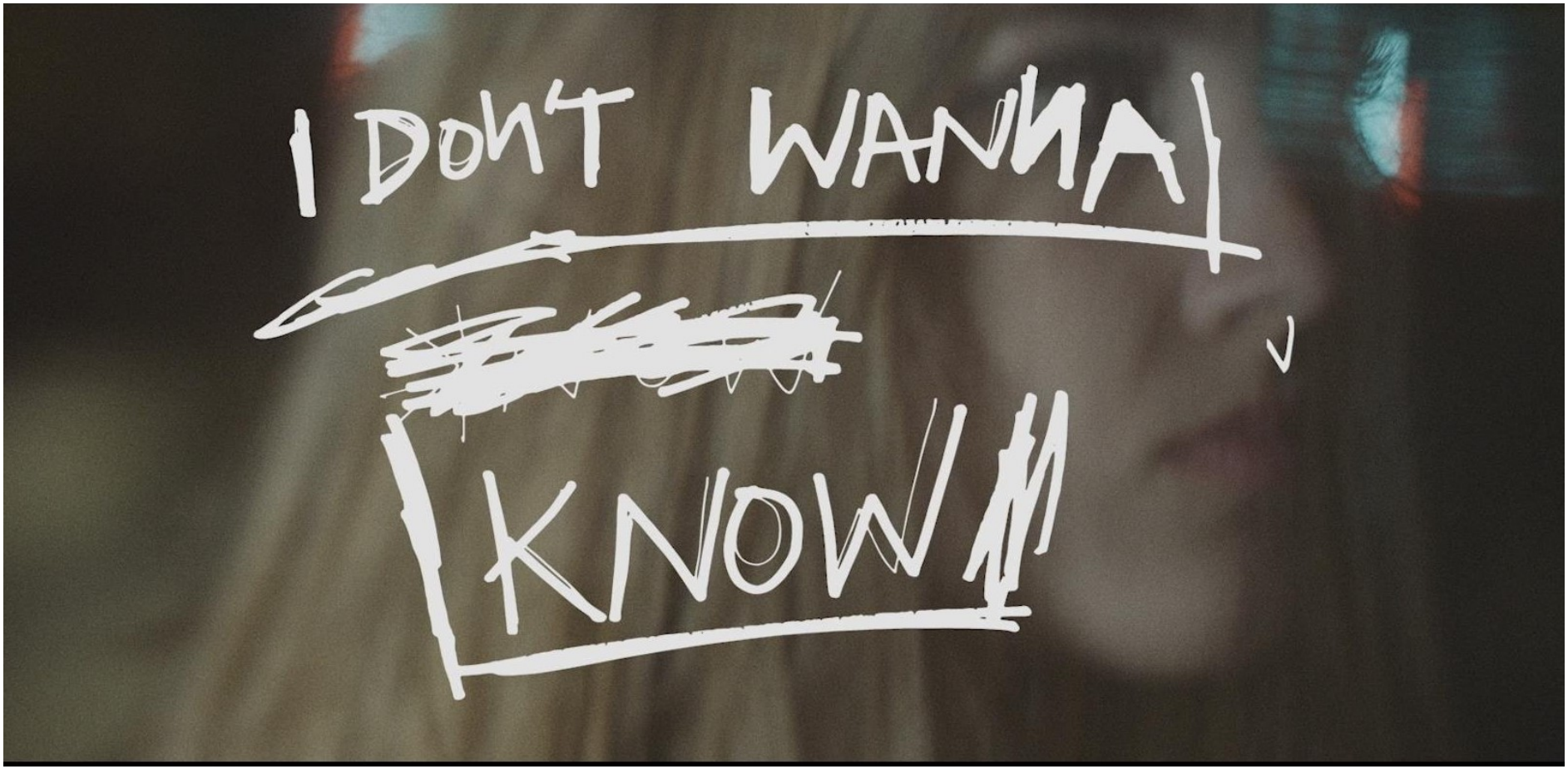
Black-box Abstraction

Why do we need the black-box?

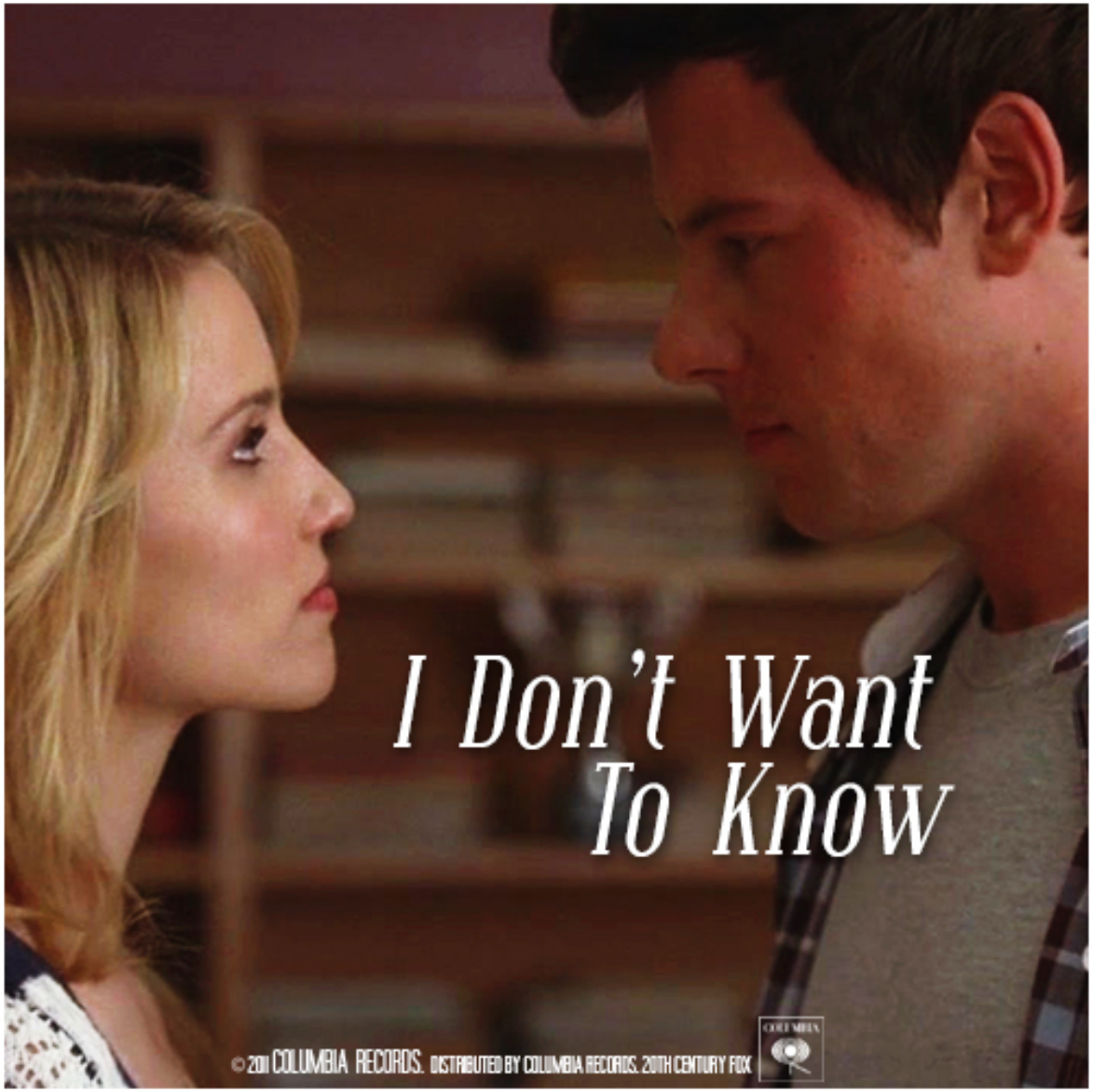
Because, for the details inside the box:

- I do not know.
- I cannot know.
- I don't want to know.
- I don't need to know.

Black-box Abstraction



Black-box Abstraction



Black-box Abstraction

You already accept this "black-box" concept!

- Do you know about the internal representation of primitives?
- But, do you use primitives?

Black-box Abstraction

Example

- The area of a circle with a radius of 3: `circle(3);`
- Why? I don't need to know!

So, what do you know?

I know:

- `circle(?);` will give me the area of a circle with a radius of ?.

Overview

To Write Good Programs

What is a good program?

- Be correct.
- Be strong.
- Be clear.
- Be beautiful.

To Write Good Programs

What is a good program?

- Be correct:
  Make sure it gives the correct answer.
- Be strong:
  Make sure it is still correct for *corner cases*.
- Be clear:
  Write proper comments to help others understand.
- Be beautiful:
  Follow the naming convention and other style requirments.

To Write Good Programs

Good coding style

- How to write comments?
- How to give names?
- Where to put whitespaces?
- Where to put line breaks?
- Where to put curly braces?
- How to use indentation?

To Write Good Programs

Example

```
// Calculates the factorial of a non-negative integer n.
function factorial(n) {
    // By definition, the factorial of 0 is 1.
    if (n === 0) {
        return 1;
    } else {
        return factorial(n - 1) * n;
    }
}

var x = 5;
factorial(2 * x);
```

To Write Good Programs

Write good programs in your submission

- In all missions and sidequests, the deduction of marks for bad coding styles may be a lot.
- A lot!
- A lot!
- A lot!
- ...

So...

- Write good programs, seriously!

Studio Group Problems

Let's discuss them now.

The End

Copyright