CSE216
Programming Abstraction

Instructor: Zhoulai Fu

State University of New York, Korea

Clarification

- Attendance checks will take place weekly
- Quizzes, or in-class assignments, will take place weekly, unless we are behind schedule.
- Each quiz taken at week N will be reviewed at week N+1.
- Next quiz will be on Wednesday, covering everything we will have learned before the quiz. Things that require hard memorization, if any, will be reminded in the quiz.
- Homework, or after-class assignments, will directly relate to coding, to be done after we have learned more about Ocaml/Java/Python

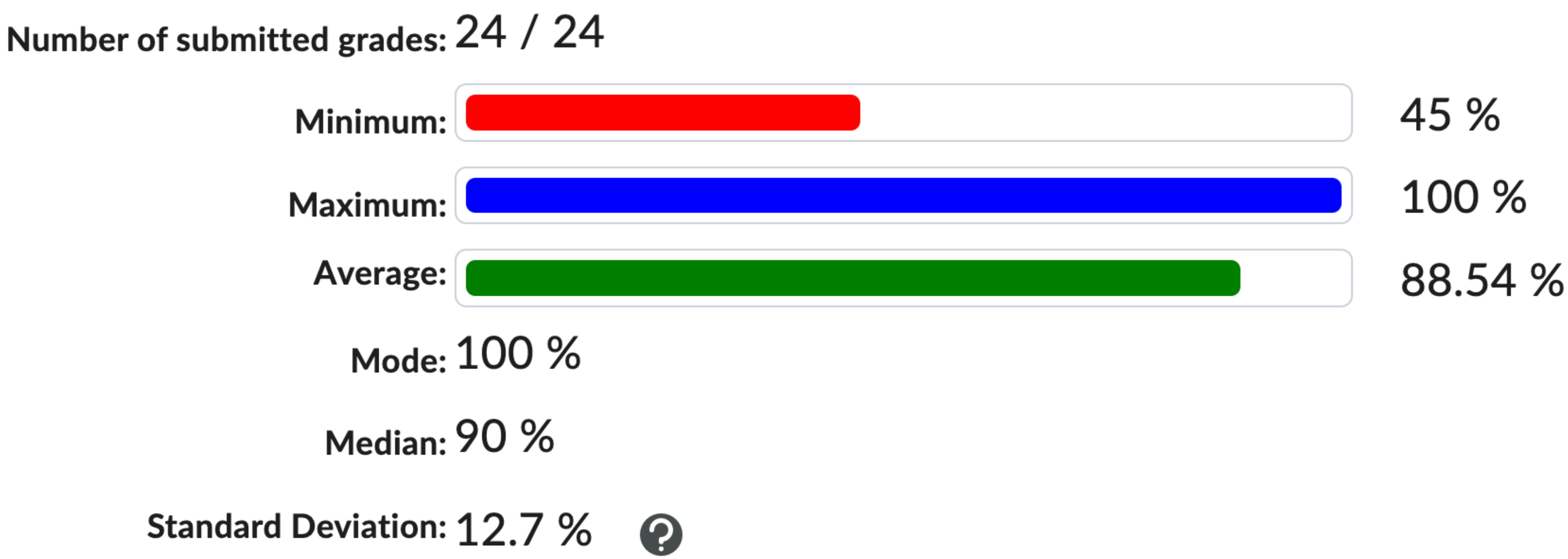
Today

- Quiz week 02 review
- Context-free grammars, derivation and ambiguity
- Lambda calculus 1

2:00-3:20

Quiz week 02 review

Quiz Week 02 statistics



- Clarification: The total score is 100.
- The last four exercises are counted as 5 points each, instead of 10.

Programming paradigms

Consider the following program in Python:

```
numbers = [1, 2, 3, 4, 5, 6]
sum = 0
for number in numbers:
    if number % 3 == 0:
        sum += number
print(sum)
```

1. (points = 10) What will be output if we run this program?

- Answer: **9**

2. (points = 10) What is the major paradigm used in the code? Choose from (a-d) below: (a) functional (b) object-oriented (c) imperative

- Answer: **c: imperative**

Lambda functions

In Python, lambda functions are defined using the lambda keyword followed by a comma-separated list of arguments (if any), followed by a colon and an expression. Here's an example:

```
add = lambda x, y: x + y
print(add(9, 3))
```

In this example, we define a lambda function `add` that takes two arguments `x` and `y`, and returns their sum. We then call the `add` function with arguments 9 and 3, which returns the sum 12.

3. (points = 10) Define an lambda function in python that takes a two input numbers and returns their distance. You can use the python function `abs` for the absolute value function. For example, `abs(-4.2)` returns 4.2. Write out the lambda expression. It should start with the key word "lambda".

- Answer: **lambda x, y: abs(x-y)**

A starter for onject-oriented programming

Consider the following code in Java

```java
public class Customer {
    private String name;
    private SeniorityLevel level;

    public Customer(String name, SeniorityLevel level) {
        this.name = name;
        this.level = level;
    }

    public String getName() {
        return name;
    }

    public SeniorityLevel getLevel() {
        return level;
    }

    public void setLevel(SeniorityLevel level) {
        this.level = level;
    }
}

enum SeniorityLevel {
    NEW, REGULAR, VIP
}
```

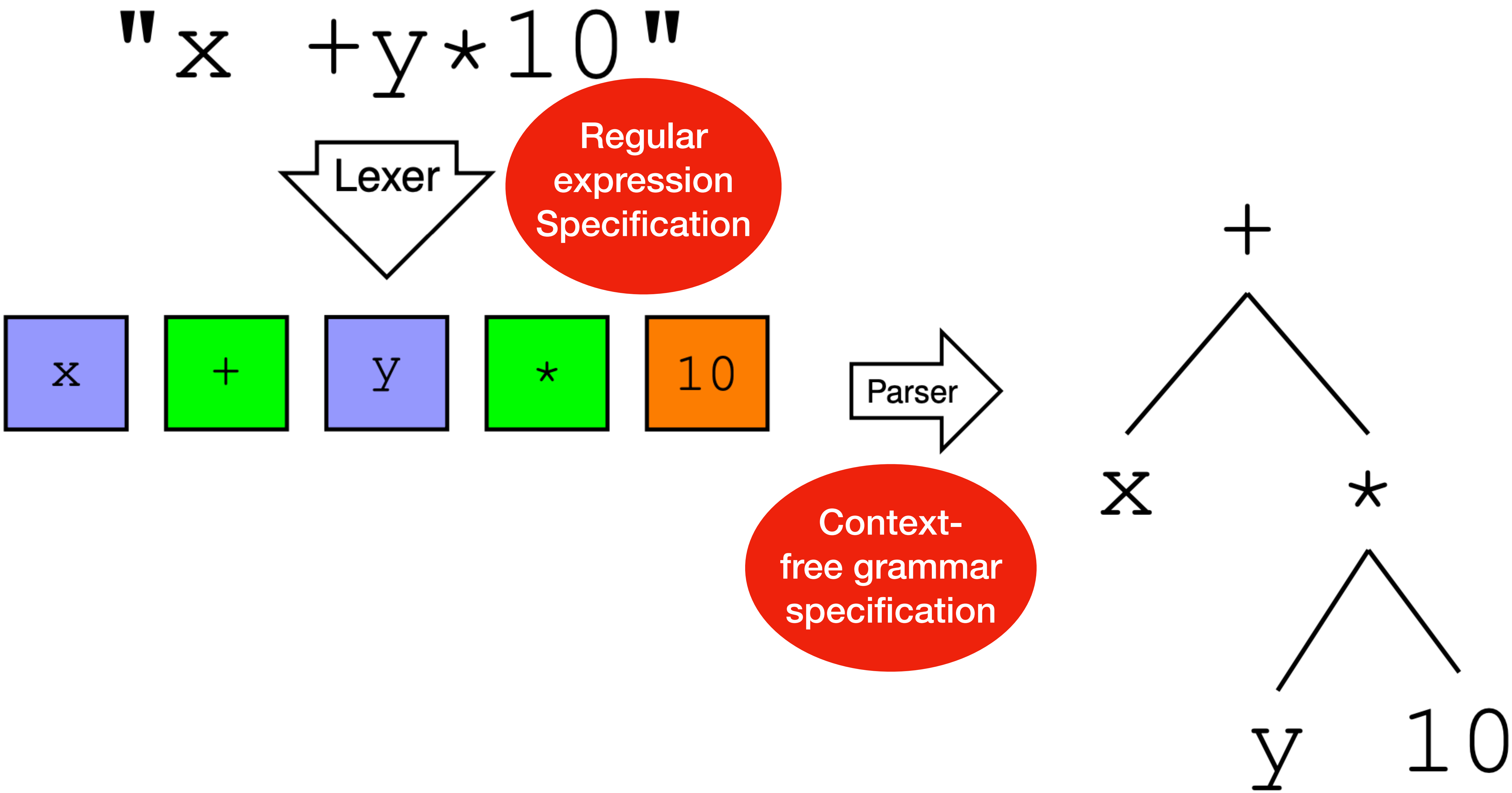
Now, your co-worker Ji-Ho is working on web development to design a frontend for customers to operate on their accounts. Assume Ji-Ho's code has access to Custom objects.

7. (points = 10) Is there any possibility that Ji-Ho's code can change a customer's name? **No**

8. (points = 10) Is there any possibility that Ji-Ho's code can change a customer's seniority level? **Yes**

Context-free grammar

Parsing


"x +y*10"
Lexer
Regular expression Specification
x
+
y
*
10
Parser
Context-free grammar specification
+
x
*
y
10

The need for a grammar

"Afternoon good, I'd room a like."

Mr. Men and Little Miss Series

Formal grammar

- Formal grammars are typically expressed using **a set of symbols and production rules** that define how those symbols can be combined to form valid sentences or expressions.
- There are **different types of formal grammars, including context-free grammars, regular grammars, and context-sensitive grammars**.
- Formal grammar is an essential tool for **analyzing and understanding the structure of languages.** It has applications in fields such as computational linguistics, natural language processing, and artificial intelligence.

Context-free grammar: Example 1

1. $S \to aSb$
2. $S \to ba$

- We start with S, and can choose a rule to apply to it.
- If we choose rule 1, we obtain the string aSb. If we then choose rule 1 again, we replace S with aSb and obtain the string aaSbb. If we now choose rule 2, we replace S with ba and obtain the string aababb, and are done.
- We can write this series of choices more briefly, using the **derivation**: S => aSb => aaSbb => aababb
- **The language of the grammar** is the infinite set {a^n ba b^n | n>0} where n is is repeated times.
- This grammar is **context-free**, with only single nonterminals appear as left-hand sides)

Example 2: arithmetic expressions (ambiguous)

```
<expr>  ::=  <expr> + <expr>
          | <expr> - <expr>
          | <expr> * <expr>
          | <expr> / <expr>
          | ( <expr> )
          | <number>
<number> ::= 0 | 1 | 2 | ... | 9
```

This grammar is **ambiguous** because it allows for multiple parse trees to represent the same expression. For example, the expression 2 + 3 * 4 can be parsed as (2 + 3) * 4 or 2 + (3 * 4)

Ambiguous grammar

- An ambiguous grammar is a formal grammar that can produce **multiple parse trees** or interpretations for the same input sentence or sequence of symbols.
- This can be problematic in various contexts because it can **make it difficult to determine the correct meaning** or parse tree of a sentence or sequence of symbols.
- To avoid ambiguity, it is often necessary to **use unambiguous grammars** or to add rules or constraints to the ambiguous grammar to disambiguate the interpretations.

Example 3: arithmetic expressions (unambiguous)

<expr> ::= <expr> + <term>

| <expr> - <term>

<term> ::= <term> * <factor>

| <term> / <factor>

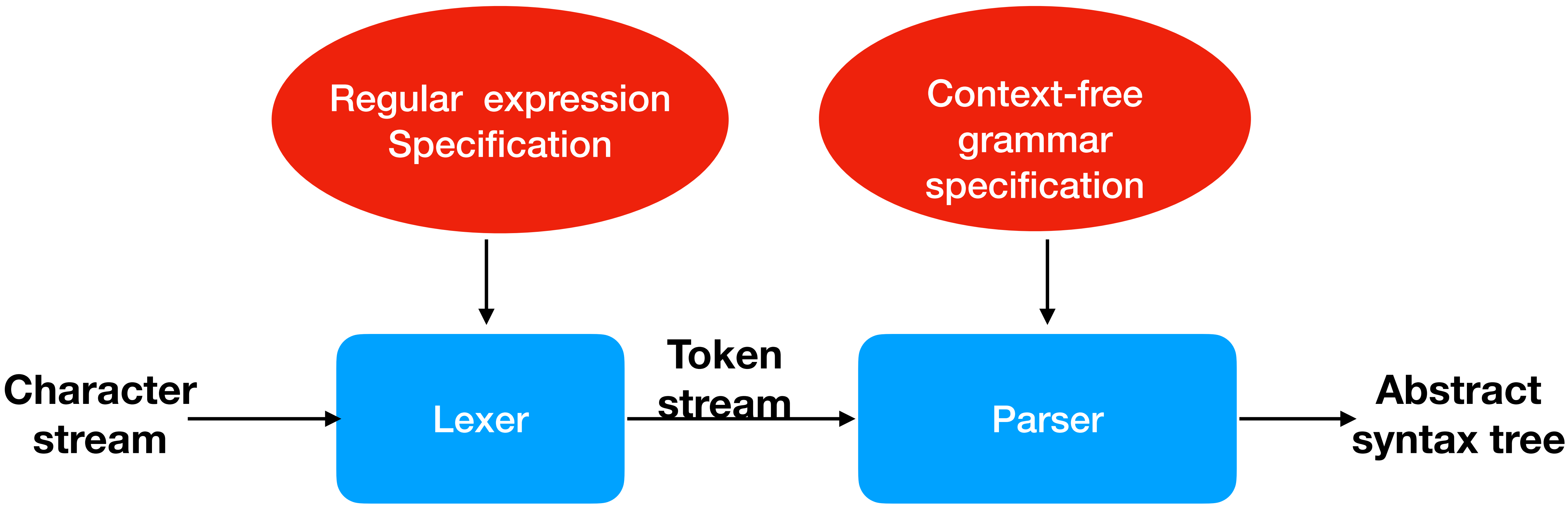
| <factor>

<factor> ::= (<expr>) | <number>

<number> ::= 0 | 1 | 2 | ... | 9

In this revised grammar, the <expr> rule now includes a <term> component, which handles multiplication and division. The <term> rule includes a <factor> component, which handles parentheses and numbers. This approach ensures that the order of operations is clear and unambiguous

Summary so far



- A lexer is a DFA transformed from a specification of regular expressions.
- A parser derives a parse tree using a non-ambiguous context-free grammar

Exercise

Give a RE for: $L = \{0^i 1^j \mid i \text{ is even and } j \text{ is odd } \}$

Solution

- $(00)^*1(11)^*$

Exercise

- Write a context-free grammar that generates the language of all strings of a's and b's that of the form a^nb^n where n>0.

Solution

$S \rightarrow 01$

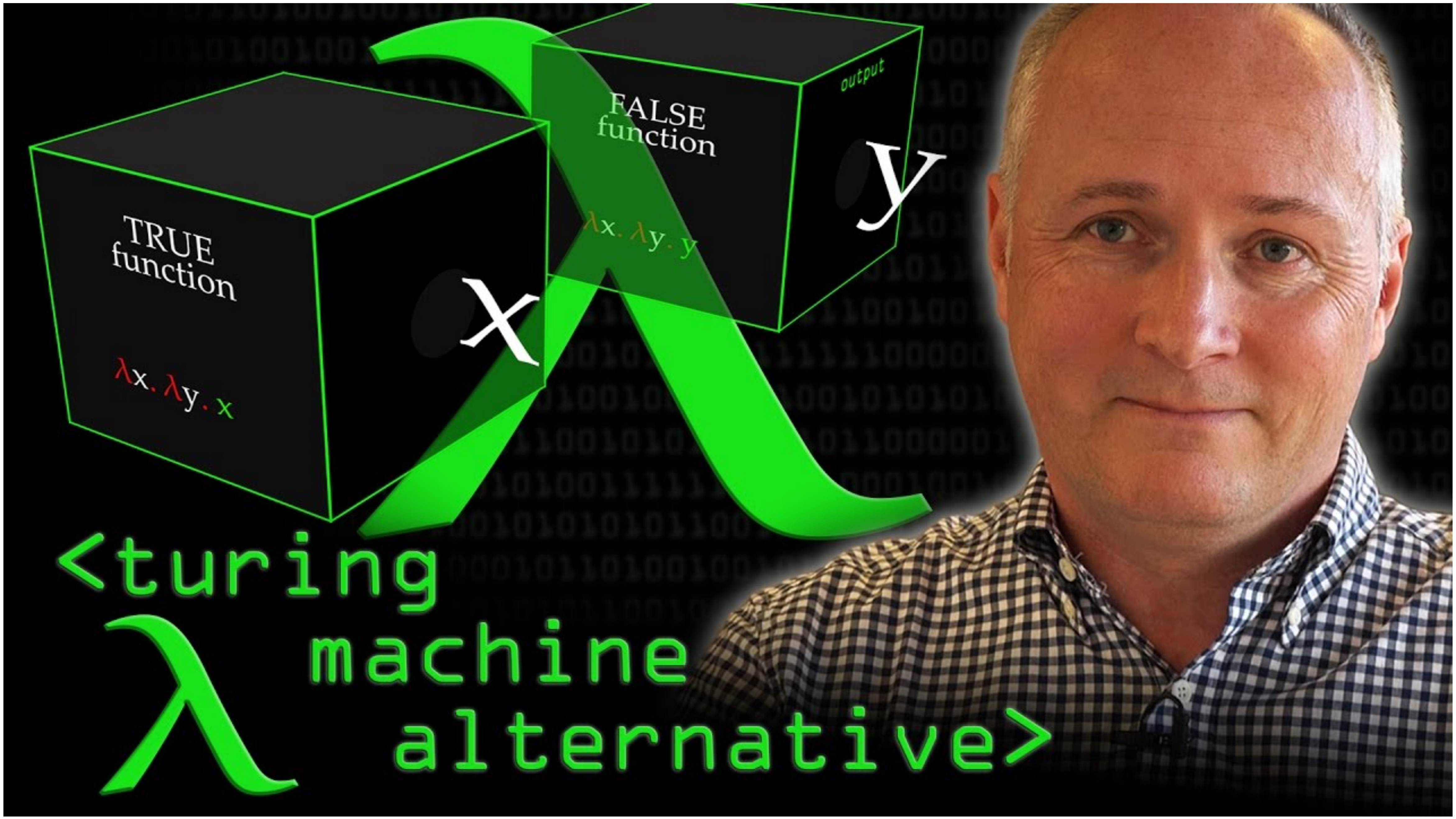
$S \rightarrow 0S1$

Lambda calculus

Many slides adapted from CMU 15-252: More Great Theoretical Ideas in Computer Science
https://www.cs.cmu.edu/~venkatg/teaching/15252-sp20/notes/lambda-calculus-slides.pdf

Thanks!

A very nice Intro to lambda calculus



https://youtu.be/eis11j_iGMs