CSE216
Programming Abstraction

Instructor: Zhoulai Fu

State University of New York, Korea

Instructor’s office hours update

- Instructor's Office hours: TU and TH 10:25 - 11:25 at B424

Today

- Quiz week 01 review
- A missing Python crash course
- Regular expressions

Quiz week 01 review

Warm up

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 = 5) What will be output if we run this program?

- Answer: **9**

2. (points = 5) What is the major paradigm used in the code? Choose a single answer from (a-e) below: (a) functional (b) object-oriented (c) imperative (d) declarative, and (e) procedural

- 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)**

Multi-Paradigm Programming Languages

JavaScript

JavaScript is a multi-paradigm programming language. For each piece of JavaScript code below, what is the major paradigm used in the code? Choose a single answer from (a-e): (a) functional (b) object-oriented (c) imperative (d) declarative, and (e) procedural

13. (points = 5)

```
let counter = 0;
for (let i = 0; i < 5; i++) {
    counter += i;
}
console.log(counter); // Output: 10
```

- Answer: **c. imperative**

14. (points = 5)

```
const numbers = [1, 2, 3, 4, 5];
const squared = numbers.map(num => num ** 2);
const evenSquared = squared.filter(num => num % 2 === 0);
const sum = evenSquared.reduce((acc, num) => acc + num, 0);
console.log(sum); // Output: 20
```

a. functional

SQL

SQL usually stands as a pure showcase of declarative programming, although vendors have extended it with procedural elements. For each piece of SQL code below, what is the major paradigm used in the code? Choose from (d-e): (d) declarative, and (e) procedural

15. (points = 5)


```sql
SELECT *
FROM Customers
WHERE PurchaseDate >= DATEADD(MONTH, -1, GETDATE());
```

- Answer: **d: declarative**

15. (points = 5)

```sql
CREATE PROCEDURE UpdateCustomerEmail
    @customerId INT,
    @newEmail NVARCHAR(255)
AS
BEGIN
    UPDATE Customers
    SET Email = @newEmail
    WHERE CustomerID = @customerId;
END;
```

Not a good exercise!

- Answer:

d & e

A missing Python crash course

- list data structure, loops, conditions, functions

- https://colab.research.google.com/drive/15eiIquB2QVacfZWadm_jIw5Xv2ihl60J#scrollTo=UhcbBQUiStHG

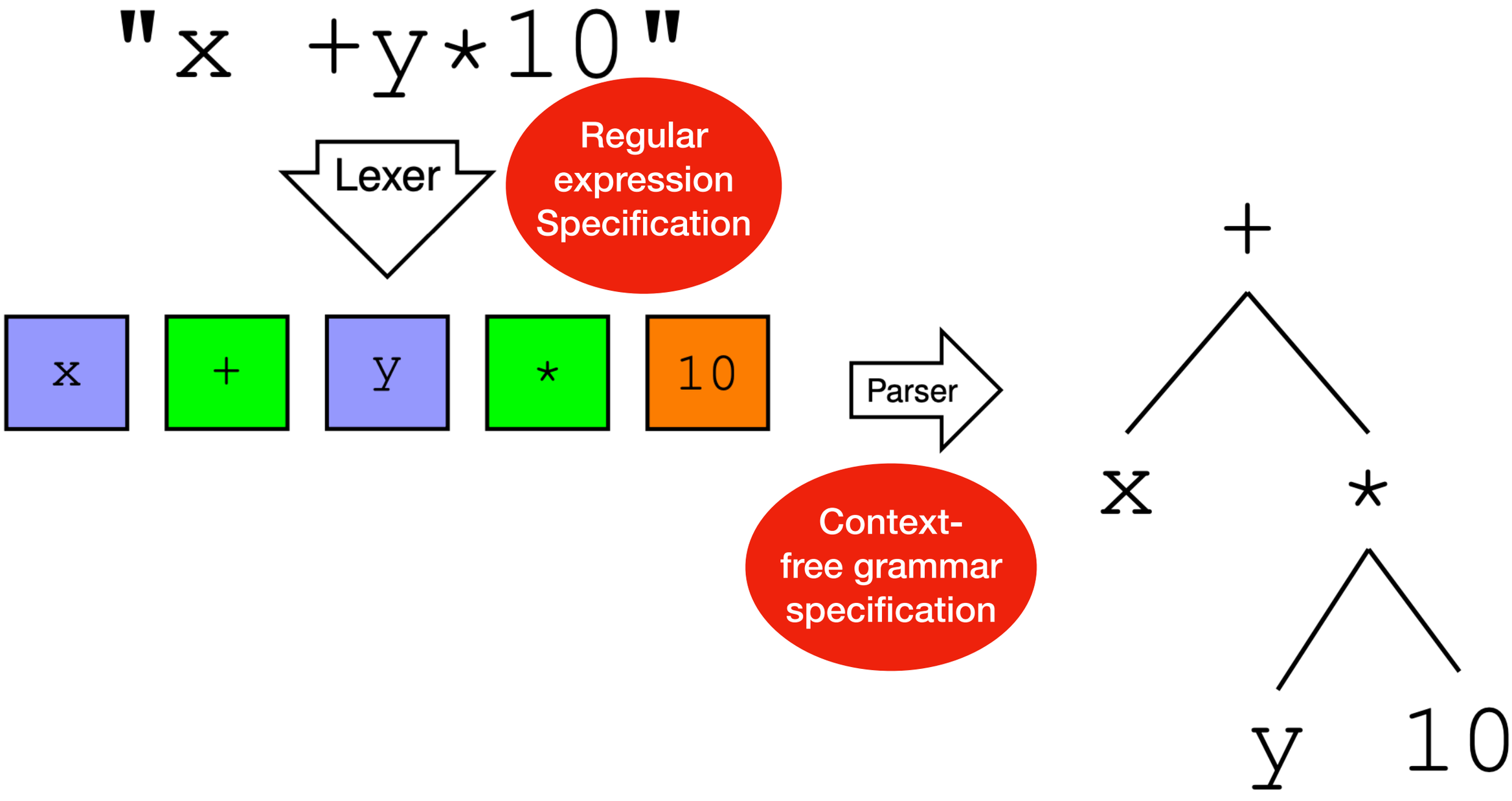
- Try to finish the exercises in the link above yourself (ungraded)

Regular Expressions

What is a program?

- Let us consider an expression x + y*10
- Think of this expression as a program in a programming language
- This is actually a program written in a programming language used by a calculator
- Today we will analyze the syntax of a general program — Syntax analysis
- Syntax analysis can take a whole semester to learn; we will touch only the surface

How to specify program syntax?



Regular expression specification looks like this in Ocaml

```
rule Token = parse
  | [' ' '\t' '\n' '\r']  { Token lexbuf }
  | ['0'-'9']+            { CSTINT (...) }
  | ['a'-'z''A'-'Z']['a'-'z''A'-'Z''0'-'9']*
                          { keyword (...) }
  | '+'                   { PLUS  }
  | '-'                   { MINUS }
  | '*'                   { TIMES }
  | '('                   { LPAR  }
  | ')'                   { RPAR  }
  | eof                   { EOF   }
  | _                     { lexerError lexbuf "Bad char" }
```

Regular expressions

r	Meaning	Language $\mathcal{L}(r)$
a	Character a	{"a"}
ε	Empty string	{""}

Regular expressions

r	Meaning	Language $\mathcal{L}(r)$
a	Character a	$\{"\mathtt{a}"\}$
ε	Empty string	$\{""\}$
$r_1 r_2$	r_1 followed by r_2	$\{s_1 s_2 \mid s_1 \in \mathcal{L}(r_1), s_2 \in \mathcal{L}(r_2)\}$

Regular expressions

r	Meaning	Language $\mathcal{L}(r)$
a	Character a	$\{"\mathtt{a}"\}$
ε	Empty string	$\{""\}$
$r_1 r_2$	r_1 followed by r_2	$\{s_1 s_2 \mid s_1 \in \mathcal{L}(r_1), s_2 \in \mathcal{L}(r_2)\}$
r^*	Zero or more r	$\{s_1 \ldots s_n \mid s_i \in \mathcal{L}(r), n \geq 0\}$

Regular expressions

r	Meaning	Language $\mathcal{L}(r)$
a	Character a	$\{"\mathtt{a}"\}$
ε	Empty string	$\{""\}$
$r_1 r_2$	r_1 followed by r_2	$\{s_1 s_2 \mid s_1 \in \mathcal{L}(r_1), s_2 \in \mathcal{L}(r_2)\}$
r^*	Zero or more r	$\{s_1 \ldots s_n \mid s_i \in \mathcal{L}(r), n \geq 0\}$
$r_1 \mid r_2$	Either r_1 or r_2	$\mathcal{L}(r_1) \cup \mathcal{L}(r_2)$

Regular expressions

r	Meaning	Language $\mathcal{L}(r)$
a	Character a	$\{"\texttt{a}"\}$
ε	Empty string	$\{""\}$
$r_1 r_2$	r_1 followed by r_2	$\{s_1 s_2 \mid s_1 \in \mathcal{L}(r_1), s_2 \in \mathcal{L}(r_2)\}$
r^*	Zero or more r	$\{s_1 \ldots s_n \mid s_i \in \mathcal{L}(r), n \geq 0\}$
$r_1 \mid r_2$	Either r_1 or r_2	$\mathcal{L}(r_1) \cup \mathcal{L}(r_2)$

Examples

$\texttt{ab}^*$ represents {"a","ab","abb",...}

$(\texttt{ab})^*$ represents {"","ab","abab",...}

$(\texttt{a}|\texttt{b})^*$ represents {"","a","b","aa","ab","ba",...}

Regular expressions

r	Meaning	Language $\mathcal{L}(r)$
`a`	Character `a`	$\{\text{"a"}\}$
ε	Empty string	$\{\text{""}\}$
$r_1 r_2$	r_1 followed by r_2	$\{s_1 s_2 \mid s_1 \in \mathcal{L}(r_1), s_2 \in \mathcal{L}(r_2)\}$
r^*	Zero or more r	$\{s_1 \ldots s_n \mid s_i \in \mathcal{L}(r), n \geq 0\}$
$r_1 \mid r_2$	Either r_1 or r_2	$\mathcal{L}(r_1) \cup \mathcal{L}(r_2)$

Examples

`ab`* represents {"a","ab","abb",...}

(`ab`)* represents {"","ab","abab",...}

(`a`|`b`)* represents {"","a","b","aa","ab","ba",...}

Exercise

What does (`a`|`b`)`c`* represent?

Regular expression abbreviations

Abbrev.	Meaning	Expansion
[aeiuo]	Set	a|e|i|o|u
[0–9]	Range	0|1|...|8|9
[0–9a–Z]	Ranges	0|1|...|8|9|a|b|...|y|z
r?	Zero or one *r*	*r*|ε
r^+	One or more *r*	*r r**

Exercise 1

Write regular expressions for:

- Non-negative integer constants

Demo: https://regex101.com/

Exercise 2

Write regular expressions for:

- Non-negative integer constants
- Integer constants

Exercise 3

Write regular expressions for:

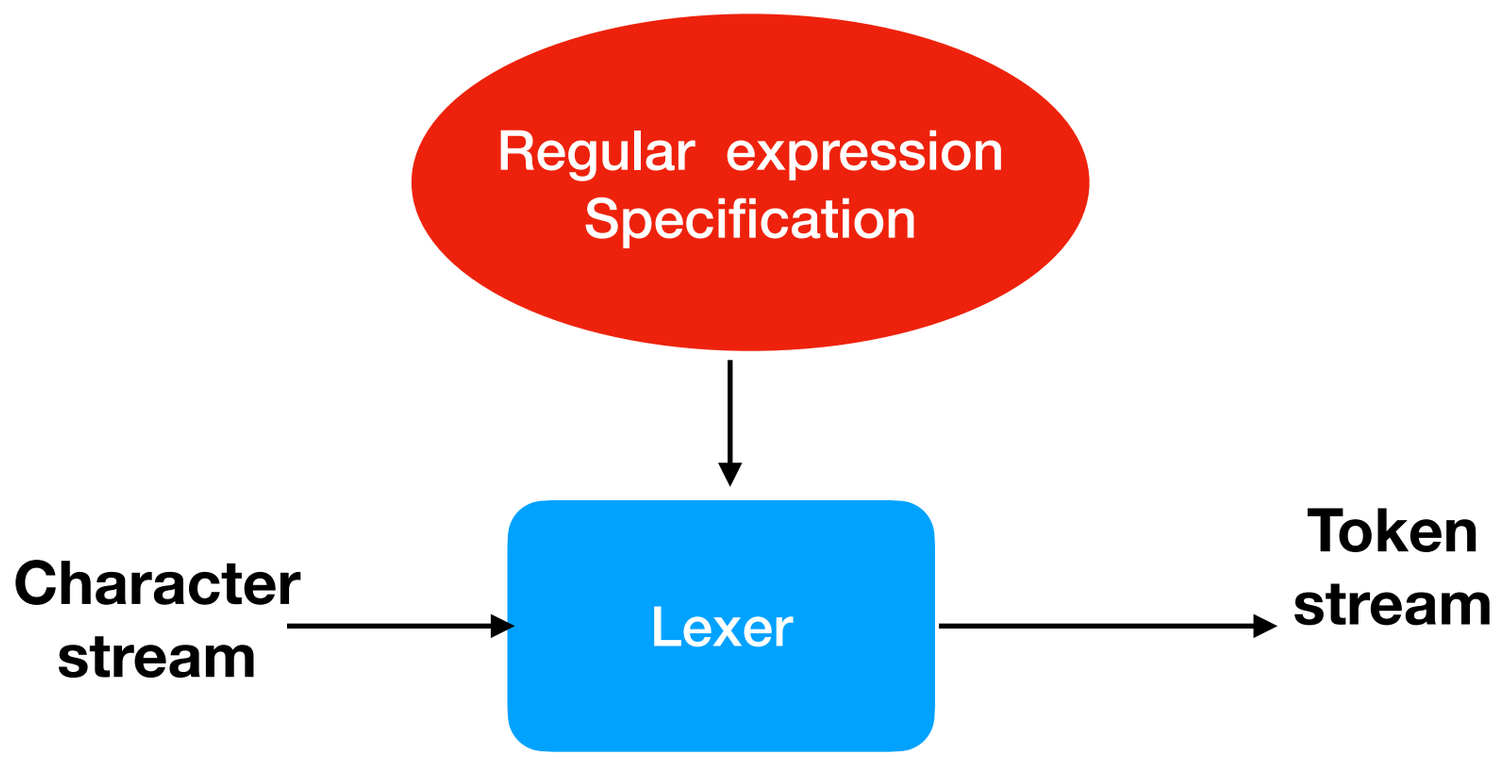
- Non-negative integer constants
- Integer constants
- Floating-point constants:
  - `3.14`
  - `3E8`
  - `+6.02E23`

Exercise 4

Write regular expressions for:

- Non-negative integer constants
- Integer constants
- Floating-point constants:
  - `3.14`
  - `3E8`
  - `+6.02E23`
- Java variable names:
  - `xy`
  - `x12`
  - `_x`
  - `$x12`

Summary



Next time: How to turn regex into a lexer?

Solutions

- Non-negative integers 0 | [1-9][0-9]*
- Integers 0| [+-]?[1-9][0-9]*
- Floating-point: [-+]?[0-9]*\.?[0-9]+([eE][-+]?[0-9]+)?
- Java variables: [a-zA-Z_][a-zA-Z0-9_]*