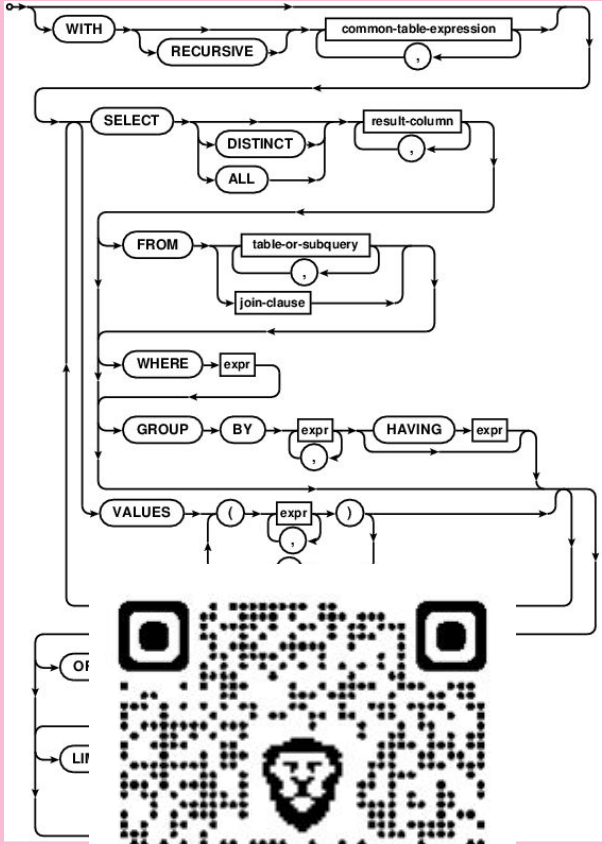


Hi! Welcome to 61A Discussion :)

We will begin at **5:10**!

Attendance: **go.cs61a.org/ben-disc**

Slides: **cs61a.bencuan.me**

Announcements

- Last discussion next week :'(
  - If you have any q's bring them in!
  - also come to lab
- Scheme exists (due Tuesday, 4/26)
  - Accepting project questions during lab!
- HW7 due **tomorrow** night (extended)

Agenda

- Attendance
- Regex
- BNF

Some quick regex tips

- Remember your r strings!
- Use regex101.com or equivalent

Using regex in python

- https://www.w3schools.com/python/python_regex.asp
- tl;dr

```
import re
re.search(r”YOUR_REGEX_HERE”, str)
```

- **r(aw) string:** every character is taken literally (no need to escape special chars)
  - **Remember to put the r in front of the string!!!**

warmup!

- Which of the following expressions will match the string “Oski”?
  - Oski
  - [Oski]
  - ([oO]ski)*
  - \w{3,6}
  - .?

Q1: Greetings

Let's say hello to our fellow bears! We've received messages from our new friends at Berkeley, and we want to determine whether or not these messages are *greetings*. In this problem, there are two types of greetings - salutations and valedictions. The first are messages that start with "hi", "hello", or "hey", where the first letter of these words can be either capitalized or lowercase. The second are messages that end with the word "bye" (capitalized or lowercase), followed by either an exclamation point, a period, or no punctuation. Write a regular expression that determines whether a given message is a greeting.

- Follow along: https://regex101.com/r/m0HuiN/1
- Match **either one OR both of:**
  - Lines that start with ‘hi’, ‘hello’, ‘hey’, ‘Hi’, ‘Hello’, ‘Hey’
  - Lines that end with ‘bye’, ‘Bye’, ‘bye!’, ‘Bye!’

Q2: Basic URL Validation

In this problem, we will write a regular expression which matches a URL. URLs look like the following:

For example, in the link `https://cs61a.org/resources/#regular-expressions`, we would have:

- Scheme: `https`
- Domain Name: `cs61a.org`
- Path to the file: `/resources/`
- Anchor: `#regular-expressions`

scheme: https:// or http://
domain name: some word characters, a dot (.), more word characters
path: any number of / followed by words followed by (word.word)
anchor: a hashtag followed by any number of words or hyphens

https://regex101.com/r/VIp5sd/1

BNF

slides adopted from sp21 review session

Why BNF??

- BNF = Backus–Naur form

- Used to define languages in real life
  - https://docs.python.org/3/reference/grammar.html
  - CS164

- A cool application of trees, regex, interpreters

Terminal Symbols

The atomic expression equivalent in BNF -- the smallest type of symbol possible.

- Serves as a **base case** for recursive rules,
- Often defined as regular expressions

```
INTEGER: /-?\d+/
STRING: /"\w+"/
```

Non-Terminal Symbols

These symbols can be recursive, and are used to define any type of syntax that this grammar might use.

```
?item: list | INTEGER | STRING
list: "[" item ("," item)* "]"
```

`?item` means that, when you visualize the syntax tree, `item` will not be included, as it only has one child and that child can be referred to directly.

Helpful Shorthands

EBNF Notation	Meaning	Pure BNF Equivalent
item*	Zero or more items	items: | items item
item+	One or more items	items: item | items item
[item] item?	Optional item	optitem: | item

Q3: What does BNF Match?

```
?start: calc_expr

?calc_expr: NUMBER | calc_op

calc_op: "(" OPERATOR
calc_expr* ")"

OPERATOR: "+" | "-" | "*" |
"/"

%ignore /\s+/
%import common.NUMBER
```

```
lark> (+ 1 2)
lark> (+)
lark> (1)
lark> (+ 1 2 3)
lark> (+ 1)
lark> (1 + 2)
lark> (+ 1 (+ 2 3))
lark> (+ 1 - 2 3)
```

Q3: What does BNF Match?

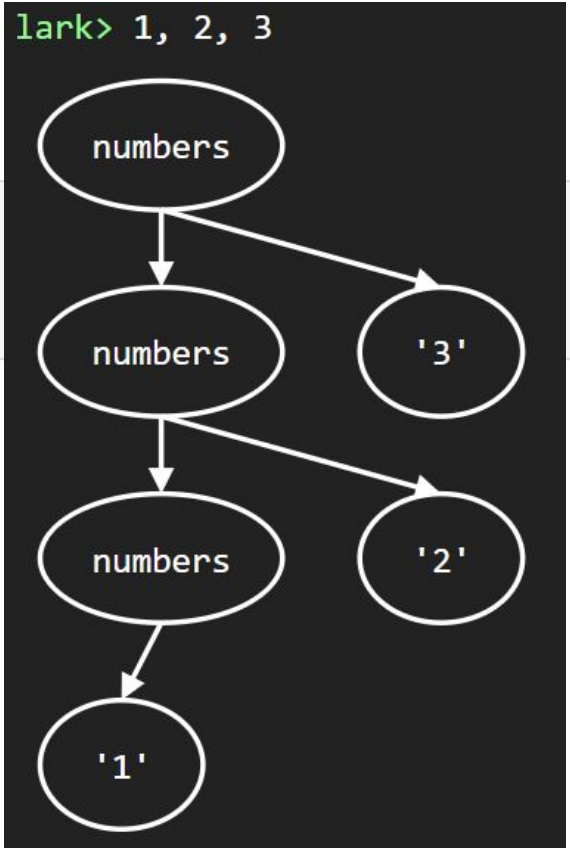
```
?start: calc_expr

?calc_expr: NUMBER | calc_op

calc_op: "(" OPERATOR
calc_expr* ")"

OPERATOR: "+" | "-" | "*" |
"/"

%ignore /\s+/
%import common.NUMBER
```

```
lark> (+ 1 2)
lark> (+)
lark> (1)
lark> (+ 1 2 3)
lark> (+ 1)
lark> (1 + 2)
lark> (+ 1 (+ 2 3))
lark> (+ 1 - 2 3)
```

Syntax Trees

```
?start: numbers
numbers: numbers "," INTEGER | INTEGER
INTEGER: /-?\d+/
%ignore /\s+/
```

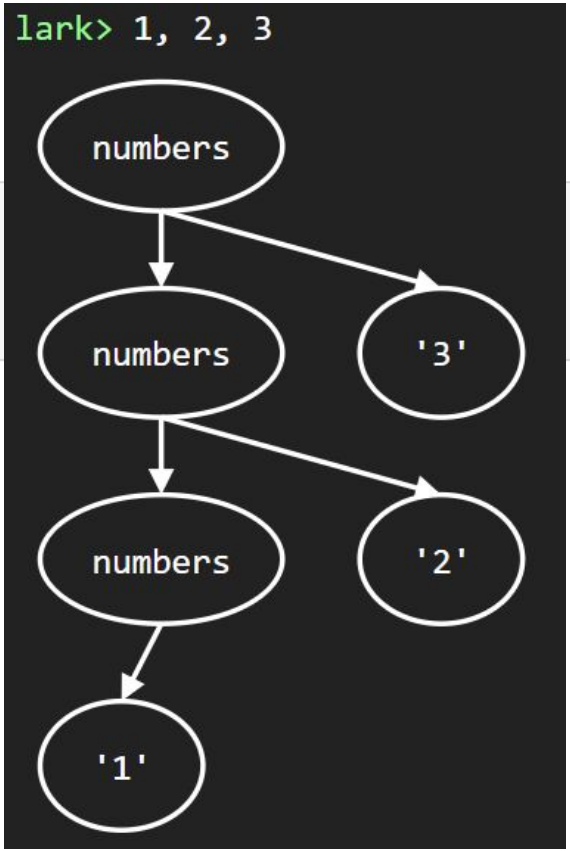
```
numbers 3
    numbers 2
        numbers 1
```



Syntax Trees

```
numbers 3
    numbers 2
        numbers 1
```

```
?start: numbers
numbers: numbers "," INTEGER | INTEGER
INTEGER: /-?\d+/
%ignore /\s+/
```



Rules:

- Ignore lines with ?
- Indent each layer
- Single leaf nodes can go on the same line as their parents

Q4: lambda BNF

```
?start: lambda_expression
lambda_expression:  "lambda " arguments ":" body
arguments: WORD ("," WORD)*
body: expression
?expression: value | lambda_expression
?value: WORD | NUMBER

%import common.WORD
%import common.NUMBER
%ignore /\s+/
```

```
lark> lambda x: 5
```

```
lark> lambda x, y: x
```

```
lark> lambda x: lambda y: x
```

Q5: Simple CSV

CSV, which stands for "Comma Separated Values," is a file format to store columnar information. We will write a BNF grammar for a small subset of CSV, which we will call SimpleCSV.

Create a grammar that reads SimpleCSV, where a file contains rows of words separated by commas. Words are characters `[a-zA-Z]` (and may be blank!) Spaces are not allowed in the file.

Here is an example of a 2-line SimpleCSV file:

```
first,second,third
fourth,fifth,sixth,,eighth
```

line: how do you encode a list of words separated by commas?

- for simplicity, you can assume at least 1 word always exists
- remember that the last word doesn't have a comma after it!

lines: how do you encode a bunch of lines together?

- the newline character is "\n"
- there may or may not be a newline at the end of the file