RegEx + BNF

Regular expressions

RegEx in the real word

Where are regular expressions used?

- Java, Perl, JS, etc.
- IDEs (e.g. VSCode)
- SQL
- Spreadsheets
- HTML

Better question: where aren't they used?

RegEx on the Web

Webpages are written with HTML tags, where each tag specifies an element on the page.

The `input` tag renders a text input field:

```
<label> Zip code
<input name="zip" type="text" pattern="\d\d\d\d\d">
</label>
```

→

Zip code

The `pattern` attribute uses a regular expression to describe what is valid for that field.

Quantifier shortcut: {n,m}

Use `{}` to specify how many instances to match.

- `{n}` matches exactly `n` instances
- `{n,}` matches `n` or more instances
- `{n,m}` matches from `n` and `m` instances

```
<label> Zip code
<input name="zip" type="text" pattern="\d{5}">
</label>
```

→

Zip code

Name That Input Pattern! #1

```
<label>TBD
<input name="tbd" type="text" pattern="[A-Za-z]{3}">
</label>
```

→

TBD

- What's a valid input?
- What's an invalid input?
- What's a good name for the field?

Name That Input Pattern! #1

```
<label>TBD
<input name="tbd" type="text" pattern="[A-Za-z]{3}">
</label>
```

→

TBD

- What's a valid input?AUS, aus
- What's an invalid input? australia, au
- What's a good name for the field? Country Code

Name That Input Pattern! #2

```
<label>TBD
<input name="tbd" type="text" pattern="\d{4}-\d{2}-\d{2}">
</label>
```

→

TBD

- What's a valid input?
- What's an invalid input?
- What's a good name for the field?

Name That Input Pattern! #2

```
<label>TBD
<input name="tbd" type="text" pattern="\d{4}-\d{2}-\d{2}">
</label>
```

→

TBD

- What's a valid input? 2020-03-13
- What's an invalid input? 2020/03/13, 03-13-2020
- What's a good name for the field? Date

Name That Input Pattern! #3

```
<label>TBD
<input name="tbd" type="text" pattern="[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,}$">
</label>
```

→

TBD

- What's a valid input?
- What's an invalid input?
- What's a good name for the field?

Name That Input Pattern! #3

```
<label>TBD
<input name="tbd" type="text" pattern="[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,}$">
</label>
```

→

TBD

- What's a valid input? someone@someplace.org
- What's an invalid input? someone@mod%cloth.co
- What's a good name for the field? Email address

RegEx Makeover! #1

Let's make a regular expression to match 24-hour times of the format `HH:MM`.

First draft: `[0-2]\d:\d\d`

- What invalid times would that match?
- How do we fix minutes?
- How do we fix hours?

Try in regexr.com!

RegEx Makeover! #1

Let's make a regular expression to match 24-hour times of the format `HH:MM`.

First draft: `[0-2]\d:\d\d`

- What invalid times would that match? 24:99
- How do we fix minutes? `[0-2]\d:[0-5]\d`
- How do we fix hours? `((2[0-3])|([0-1]\d)):[0-5]\d`

Try in regexr.com!

RegEx Makeover! #2

Let's make a regular expression to match any tweet talking about GME stock.

First draft: `GME`

- Would that match any non-GME tweets?

- How do we match only GME?

Try in regexr.com!

RegEx Makeover! #2

Let's make a regular expression to match any tweet talking about GME stock.

First draft: `GME`

- Would that match any non-GME tweets? Yes, like #HUGME or #HUGMEHARDER
- How do we match only GME? `\bGME\b`

Try in regexr.com!

BNF

BNF for Toddler-ese

```
start: sentence
sentence: describe_wants | describe_feeling
describe_wants: TODDLER "wants" noun_phrase "!"
noun_phrase: ARTICLE? NOUN
describe_feeling: TODDLER "is" EMOTION "!"

TODDLER: "beverly" | "baggy" | "you"
ARTICLE: "the" | "a" | "an" | "un" | "una"
NOUN: "ball" | "elmo" | "chalk" | "gusano"
EMOTION: "sad" | "mad" | "tired"

%ignore /\s+/
```

What sentences can that parse?

Try in code.cs61a.org!

BNF in the real word

Where is BNF used?

- Language specification: Python, CSS, SaSS, XML
- File formats: Google's robots.txt
- Protocols: Apache Kafka
- Parsers and compilers
- Text generation

You will likely use your BNF *reading* skills more than your BNF *writing* skills.

BNF for Calculator

```
start: calc_expr

?calc_expr: NUMBER | calc_op

calc_op: "(" OPERATOR calc_expr* ")"

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

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

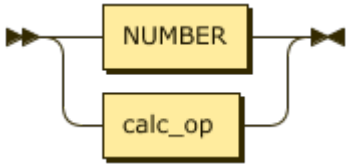
What expressions can that parse?

Try in code.cs61a.org!

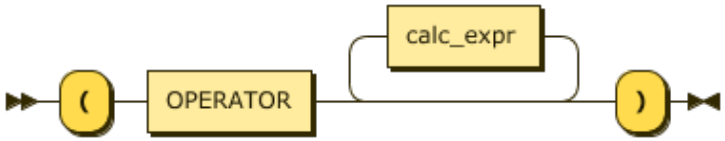
Syntax diagram for Calculator

A syntax diagram is a common way to represent BNF & other context-free grammars. Also known as railroad diagram.

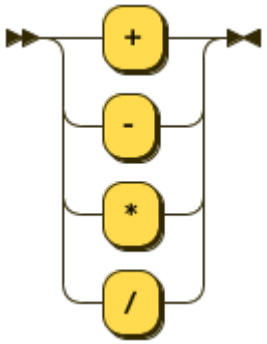
```
calc_expr: NUMBER | calc_op
```



```
calc_op: '(' OPERATOR calc_expr* ')'
```



```
OPERATOR: '+' | '-' | '*' | '/'
```



BNF for Python Integers

Adapted from the Python docs:

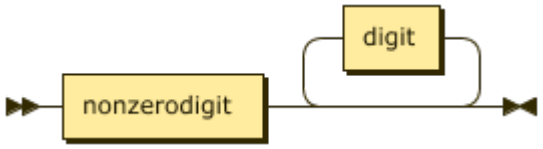
```
?start: integer
integer:  decinteger | bininteger | octinteger | hexinteger
decinteger:  nonzerodigit digit*
bininteger:  "0" ("b" | "B") bindigit+
octinteger:  "0" ("o" | "O") octdigit+
hexinteger:  "0" ("x" | "X") hexdigit+
nonzerodigit:  /[1-9]/
digit:  /[0-9]/
bindigit:  /[01]/
octdigit:  /[0-7]/
hexdigit:  digit | /[a-f]/ | /[A-F]/
```

What number formats can that parse?

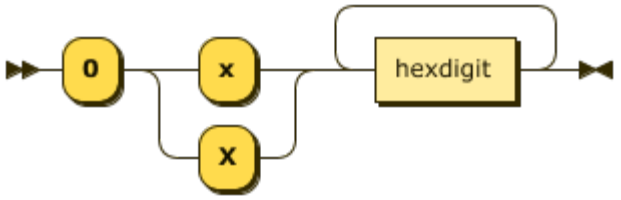
Try in code.cs61a.org!

Syntax diagram: Python numbers

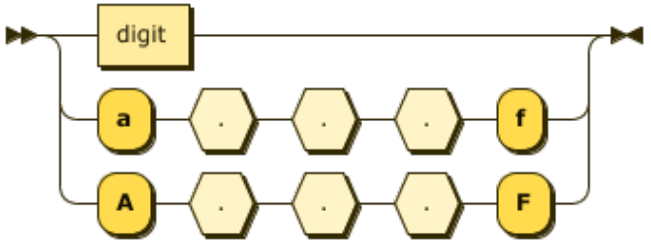
```
decinteger: nonzerodigit digit*
```



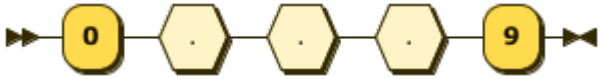
```
hexinteger: "0" ("x" | "X") hexdigit+
```



```
hexdigit: digit | /[a-f]/ | /[A-F]/
```



```
digit: /[0-9]/
```



BNF for Scheme expressions

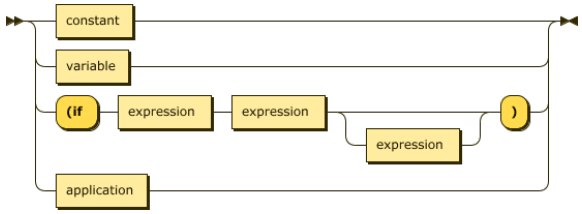
Adapted from the Scheme docs:

```
?start: expression
expression: constant | variable | "(if " expression expression expression? ")" | application
constant: BOOLEAN | NUMBER
variable: identifier
application: "(" expression expression* ")"

identifier: initial subsequent* | "+" | "-" | "..."
initial: LETTER | "!" | "$" | "%" | "&" | "*" | "/" | ":" | "<" | "=" | ">" | "?" | "~" | "_" | "^"
subsequent: initial | DIGIT | "." | "+" | "-"
LETTER: /[a-zA-z]/
DIGIT: /[0-9]/
BOOLEAN:  "#t" | "#f"

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

*This BNF does not include many of the special forms, for simplicity.

Syntax diagram: Scheme expressions

```
expression: constant | variable | "(if " expression
expression expression? ")" | application
```



```
application: "(" expression expression* ")"
```

```
identifier: initial subsequent* | "+" | "-" | "..."
```