Lab 12 Solutions lab12.zip (lab12.zip)

Solution Files

Topics

Consult this section if you need a refresher on the material for this lab. It's okay to skip directly to the questions and refer back here should you get stuck.

Questions

Regular Expressions

Q1: Calculator Ops

Write a regular expression that parses strings written in the 61A Calculator language and returns any expressions which have two numeric operands, leaving out the parentheses around them.

```
import re

def calculator_ops(calc_str):
    """
    Finds expressions from the Calculator language that have two
    numeric operands and returns the expression without the parentheses.

    >>> calculator_ops("(* 2 4)")
    ['* 2 4']
    >>> calculator_ops("(+ (* 3 (+ (* 2 4) (+ 3 5))) (+ (- 10 7) 6))")
    ['* 2 4', '+ 3 5', '- 10 7']
    >>> calculator_ops("(* 2)")
    []
    """
    # Since hyphen is a special character inside [], it must be escaped
    return re.findall(r"\(([+\-/*]\s+\d+\s+\d+)\)", calc_str)

    # Alternate solution: hyphen must be at either beginning or end inside []
    return re.findall(r"\(([-+*/]\s+\d+\s+\d+)\)", calc_str)
```

Use Ok to test your code:

```
python3 ok -q calculator_ops
```

BNF

Q2: Calculator BNF

Consider this BNF grammar for the Calculator language:

```
?start: calc_expr

?calc_expr: NUMBER | calc_op

calc_op: "(" OPERATOR calc_expr* ")"

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

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

Let's understand and modify the functionality of this BNF with a few questions.

Use Ok to test your understanding:

```
python3 ok -q ebnf-grammar-wwpd -u
```

Q3: Linked List BNF

For the next two problems, you can test your code on code.cs61a.org (https://code.cs61a.org/) by adding the following line at the beginning before the problem's skeleton code:

```
?start: link
-- replace link with tree_node for the next question
```

In this problem, we're going to define a BNF that parses integer Linked Lists created in Python. We won't be handling `Link.empty`.

For reference, here are some examples of Linked Lists:

Your implementation should be able to handle nested Linked Lists, such as the third example below.

- `Link(2)`
- `Link(12, Link(2))`
- `Link(5, Link(7, Link(Link(8, Link(9)))))`

```
link: "Link(" link_first link_rest? ")"

?link_first: link|NUMBER

?link_rest: ", " link

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

Use Ok to test your code:

```
python3 ok -q linked_list
```

Q4: Tree BNF

Now, we will define a BNF to parse Trees with integer leaves created in Python.

Here are some examples of Trees:

Your implementation should be able to handle Trees with no branches and one or more branches.

- `Tree(2)`
- `Tree(6, [Tree(1), Tree(3, [Tree(1), Tree(2)])])`

```
tree_node: "Tree(" label branches? ")"

?label: NUMBER

branches:", [" (tree_node ",")* tree_node "]"

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

Use Ok to test your code:

```
python3 ok -q tree
```