UPPSALA UNIVERSITET

MA2: A calculator

Tom Smedsaas

2021-09-06

Demonstration

```
Numerical calculator version 2021-09-07
Input : 1  - (5-2*2)/(1+1) - (-2+1)
Result: 1.5
Input : sin(3.14159265)          # Standard functions
Result: 3.5897930298416118e-09
Input : cos(PI)                  # Predefined constant
Result: -1.0
Input : 2*sin(PI/2) + log(exp(4/2 - 1))
Result: 3.0
Input : 1+2+3+4 = x              # Variable. Assignment to the right
Result: 10.0
Input : x/2 + x
Result: 15.0
Input : (1=x) + sin(2=y)         # Assignments have values
Result: 1.9092974268256817
Input : (((x + y)))
Result: 3.0
```

Demonstration continued

```
Input : ans     # Predefined variable containing last computed value
Result: 3.0
Input : 2*ans + 1
Result: 7.0
Input : 4=y=z                    # Multiple assignment
Result: 4.0
Input : vars                     # Built in command
  E     : 2.718281828459045
  PI    : 3.141592653589793
  ans   : 4.0
  x     : 1.0
  y     : 4.0
  Z     : 4.0
```

Demonstration continued

```
Input : max(sin(x), cos(x), 0.5)
Result: 0.8414709848078965
Input : fib(6)
Result: 8                    # Note integer result!
Input : fib(200)
Result: 280571172992510140037611932413038677189525
Input : fac(50)
Result: 30414093201713378043612608166064768844377641568960512000000000000
Input : file test            # Read lines from  file
File  : sin(x*x) + cos(x*x)
Result: 1.0
File  : log(E)
Result: 1.0
File  : exp(log(PI)) - PI
Result: 0
Input:                       # Back to standard input
```

Demonstration of error handling

```
Input : 1 = 2
*** Syntax error:  Expected name after '='
Error occurred at '2' just after '='
Input : 1++
*** Syntax error: Expected number, word or '('
Error occurred at token '+' just after token '+'
Input : 1+2 xxx
*** Syntax error:  Expected end of line or an operator
Error ocurred at 'xxx' just after '2'
Input : log(2 + 3 – 7)
*** Evaluation error:  Illegal argument: 'log(-2.0)'
Input : fib(3.5)
*** Evaluation error:  Argument to fib is 3.5. Must be integer >= 0
Input : fib(3.0)          # Note that this works!
Result: 2
```

UPPSALA UNIVERSITET

What do you need to know to solve the task?

- A parsing technique called recursive descent
- How to use a tokenizer
- Error handling with exceptions
- Handling function objects

Interpreting of expressions

How do we evaluate $a + (b - 1)*d - e*(f + g*h)/4$?

1. $t_1 = b - 1$
2. $t_2 = t_1*d$
3. $t_3 = g*h$
4. $t_4 = f + t_3$
5. $t_5 = e*t_4$
6. $t_6 = t_5/4$
7. $t_7 = a + t_2$
8. $t_8 = t_7 - t_6$

A structured description

$$a + (b - 1)*d - e*(f + g*h)/4$$

Three *terms*:

a

$(b - 1)*d$

$e*(f + g*h)/4$

The terms should be calculated and then summed/subtracted

A structured description

Each *term* consists of *factors*.

For example, the term $e * (f + g*h)/4$ consists of three factors:

e	a *variable*
$(f + g * h)$	an *expression*
4	a *constant*

These should be calculated and then multiplied or divided

Structured description

Summary

An *expression* is a sequence of one ore more *terms* separated by + or –

A *term* is a sequence of one or more *factors* separated by * or /

A factor is a *number*, a *variable* or an *expression enclosed by parenthesis*

Note the recursion!

Simplified expressions

The program you download can only handle this type of expressions:

An *expression* is a sequence of one or more *terms* separated by +

A *term* is a sequence of one or more *factors* separated by *

A *factor* is a *number* or an *expression* enclosed by parenthesis.

Syntax charts and pseudo code

An expression is sequence of terms separated by +

```
def expression():
    sum = term()
    while # next is ‘+’:
        # Get passed ‘+’
        sum += term()
    return sum
```

Syntax charts and pseudo code

A term is sequence of factors separated by *

```
def term():
    prod = factor()
    while # next is '*':
        # Get passed '*'
        prod *= factor()
    return prod
```

Syntax charts and pseudo code

A factor is either a number or an expression enclosed by parenthesis

```
def factor():
    if # next is "(":
        # Get passed "("
        result = expression()
        # Get passed ")"
    else:
        result = # read number
    return result
```

Some things to think about

- We mix reading of numbers and characters – don't know what is coming.
- The code shall ignore spaces.
- We have no syntactic description of numbers.
- How do we know that an expression is complete?
- What are the base case(es) in the recursion?
- What happens if we enter illegal expressions?

UPPSALA UNIVERSITET

A tokenizer

- The input to the program is a stream of characters but we want our program to work with tokens like numbers, words and, sometimes, characters.
- A tokenizer puts together characters into *tokens* which are the units we want to handle.
- In a general tokenizer we one want to specify the rules for how to tokens together.

Your tokenizer

We provide an interface class `TokenizeWrapper` to the standard module `tokenize`

TokenizeWrapper

Method	Function
`TokenizeWrapper(line)`	
`next()`	Advances to next token
`has_next()`	`True` if more tokens, else `False`
`get_current()`	Returns the current token as a string
`get_previous()`	Returns the previous token as a string
`is_number()`	`True` if current token is a number, else `False`
`is_name()`	`True` if current token is a name
`is_at_end()`	`True` if at end of line (EOL) else `False`

Note: Only next() advances! The other methods deliver info.

Usage in the parser

```
def expression(wtok):
    result = term(wtok)
    while wtok.get_current() == "+":
        wtok.next()                    # bypass +
        result = result + term(wtok)
    return result
```

A main function

```
def main():
    print("Very simple calculator")
    while True:
        line = input("Input : ")
        wtok = TokenizeWrapper(line)
        if wtok.get_current() == "quit":
            break
        else:
            print("Result: ", expression(wtok))
    print("Bye!")
```

How to start?

- Read the MA2 text
- Download the the files `MA2.py` and `MA2tokenizer.py`
- Also download the the `MA2test.txt` but it is of no use until you have implemented the `file` command.
- Run the `MA2.py` and check that addition, multiplication and parenthesis works.
- Modify the code so that subtraction and division are handled.
- …

Syntax

assignment
expression
=
name
expression
term
+ or −
term
factor
* or /
arglist
(
assignment
)
,
factor
(
assignment
)
function_1
(
function_n
arglist
name
number
−
factor

The end

I will record another lecture where I discuss

- Error handling with exceptions
- Function objects
- And more details.