UPPSALA UNIVERSITET

MA2: A calculator

- Introduction to the second assignment.
- Description of *recursive descent.*
- Presentation of a *tokenizer.*
- Presentation of a very small demonstration program.

UPPSALA UNIVERSITET

Demonstration

```
Numerical calculator version 2022-03-10
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: 1.909297426825681 7
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
```

UPPSALA UNIVERSITET

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*

UPPSALA UNIVERSITET

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. $t_8 = t_7 - t_6$

UPPSALA UNIVERSITET

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

UPPSALA UNIVERSITET

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

UPPSALA UNIVERSITET

Structured description

Summary

An *expression* is a sequence of one or 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!

UPPSALA UNIVERSITET

Simplified expressions

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

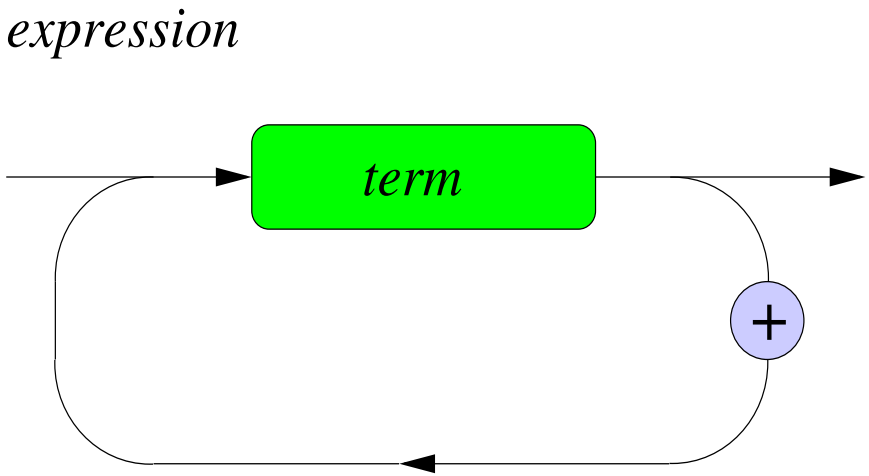
An *expression* is a sequence of one or more *terms* separated by +.

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 +

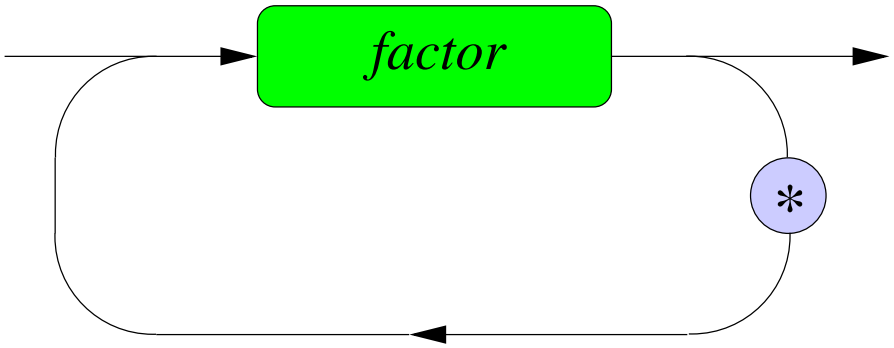



```
def expression():
    sum = term()
    while # next is ‘+’:
        # Get passed ‘+’
        sum += term()
    return sum
```

Syntax charts and pseudo code

A term is sequence of factors separated by *

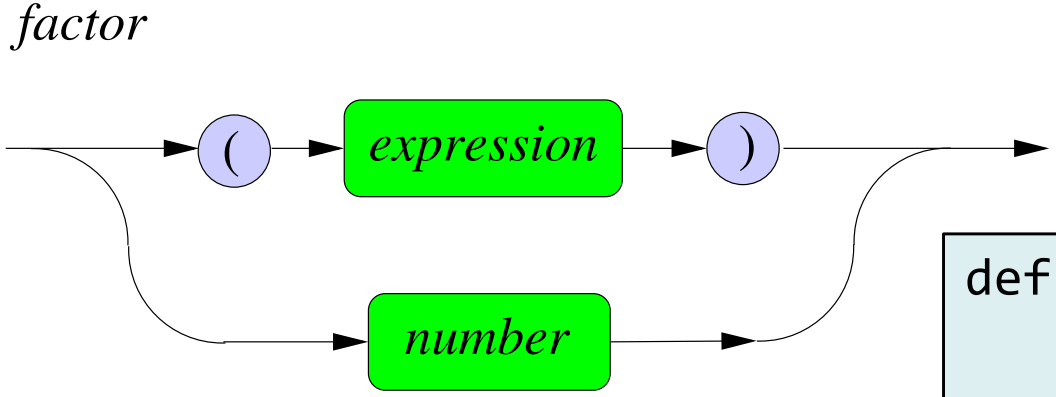
term




```
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
```

UPPSALA UNIVERSITET

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

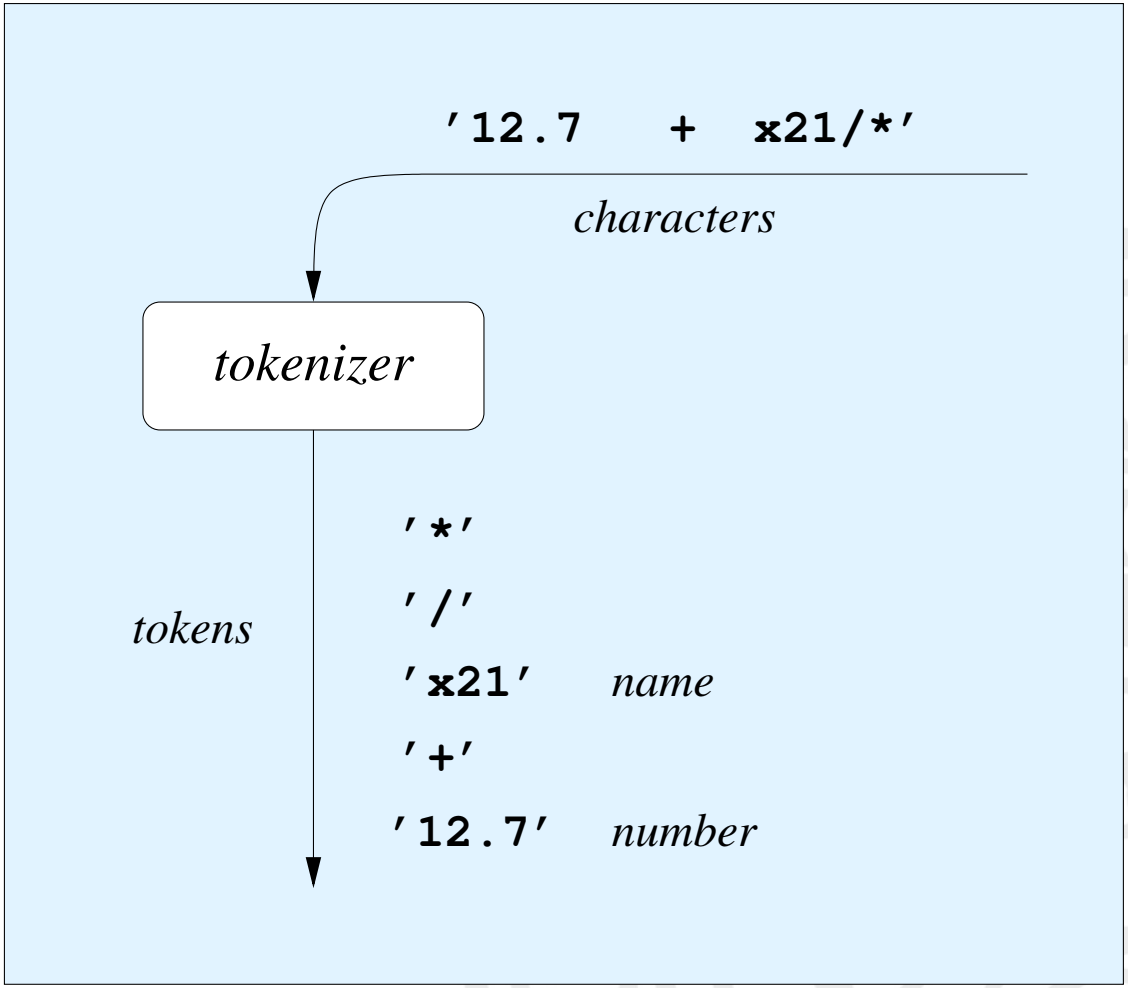
A tokenizer

- 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.

UPPSALA UNIVERSITET

Your tokenizer

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



UPPSALA UNIVERSITET

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.

UPPSALA UNIVERSITET

A `main` function

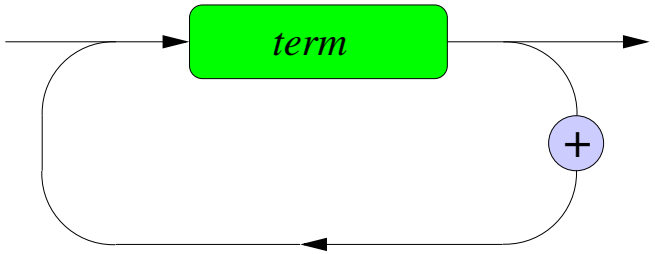
```
def main():
    print("Very simple calculator")
    while True:
        line = input("Input : ")
        wtok = TokenizeWrapper(line)
        if wtok.get_current() == "quit":
            break
        else:
            print("Result: ", expression(wtok))
    print("Bye!")
```

The tokenizer is needed in all functions so it is passed as an argument.

UPPSALA UNIVERSITET

Usage in the parser

expression




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

Since all functions need access to the tokenizer it is passed as an argument to all parser functions.

UPPSALA UNIVERSITET

A `main` function

- and so on in *term* and *factor.*
- This tiny program is available in the downloaded file `MA2micro.py`. It can be used for experiments.
- In the next video I will discuss the real program.

The end

UPPSALA UNIVERSITET

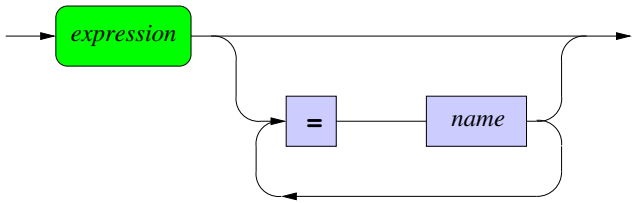
How to start?

- 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.
- …

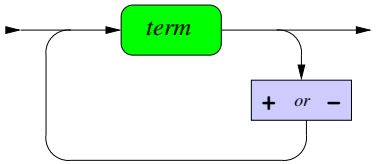
UPPSALA UNIVERSITET

Syntax

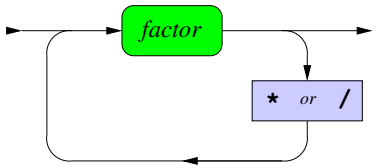
assignment



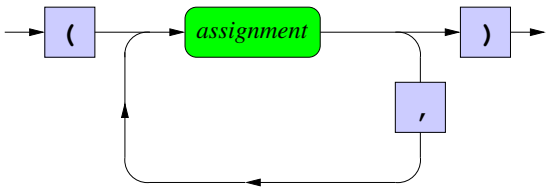
expression



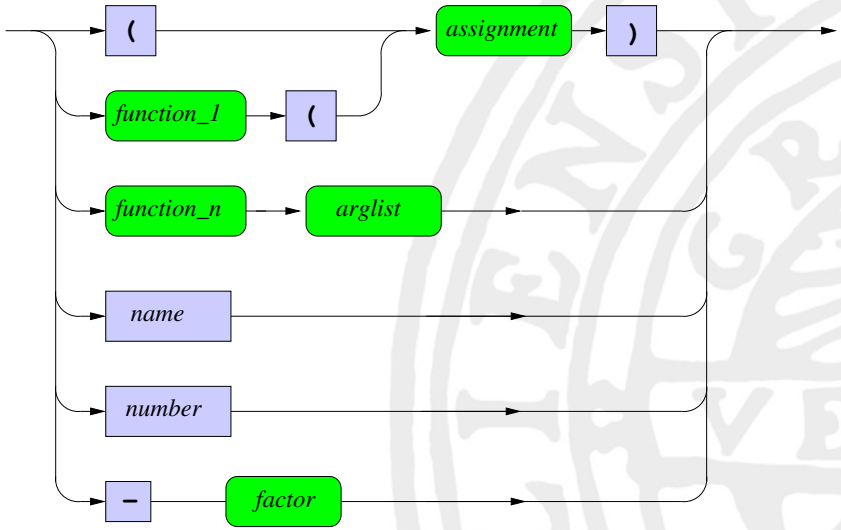
term



arglist



factor



UPPSALA UNIVERSITET

The end

I will record another lecture where I discuss

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

UPPSALA UNIVERSITET

Undantag (exceptions)

En generell mekanism för att hantera ”fel”.

- Används automatiskt av Python för olika typer av fel som kan inträffa när programmet körs.
- Kan användas av programmeraren för att hantera fel som programmet upptäcker.

UPPSALA UNIVERSITET

Exempel

Så här ser det ut om vi i Python-konsolen dividerar med 0:

```
>>> 1/0
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
ZeroDivisionError: division by zero
```

Andra exempel på undantag:

`math.sin('a')` skapar `TypeError`

`math.log(-1)` skapar `ValueError`

UPPSALA UNIVERSITET

Undantag kan ”fångas”

```
from math import sqrt


while True:
    x = input("Give a positive number: ")
    try:
        y = sqrt(int(x))
    except ValueError:
        print(f"Squarehead! {x} is not a positive number!")
    else:
        print(f"The square root of {x} is {y}")
```

UPPSALA UNIVERSITET

Körresultat

```
Give a positive number: 2
The square root of 2 is 1.4142135623730951
Give a positive number: -1
Squarehead! -1 is not a positive number!
Give a positive number: 4
The square root of 4 is 2.0
Give a positive number: a
Squarehead! a is not a positive number!
Give a positive number: 9
The square root of 9 is 3.0
Give a positive number:
```

UPPSALA UNIVERSITET

Exempel

```
from math import sqrt, log

def compute(x):
    return sqrt(x) + log(2-x) + 1/x

while True:
    x = input('Give a value: ')
    try:
        y = compute(int(x))
    except (ValueError,
            ZeroDivisionError) as e:
        print('*** Error:', e)
    else:
        print('Result: ', y)
```

UPPSALA UNIVERSITET

Exempelkörning

```
Give a value: 1
Result:  2.0
Give a value: 2
*** Error: math domain error
Give a value: -2
*** Error: math domain error
Give a value: a
*** Error: invalid literal for int() with base 10: 'a'
Give a value: 0
*** Error: division by zero
Give a value:
```

UPPSALA UNIVERSITET

Hantera fel på olika sätt

```
from math import sqrt, log

def compute(x):
    return sqrt(x) + log(2-x) + 1/x

while True:
    x = input('Give a value: ')
    try:
        y = compute(int(x))
    except ValueError as e:
        print('*** Illegal argument!', e)
    except ZeroDivisionError as e:
        print('*** Division by zero!', e)
    else:
        print('Result: ', y)
```

UPPSALA UNIVERSITET

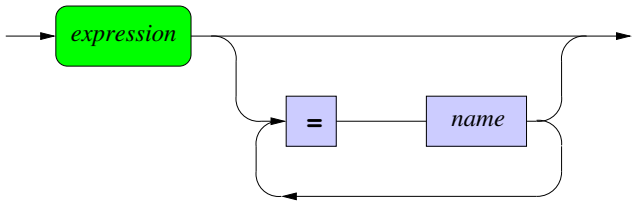
Egna undantagsklasser

```
class MyException(Exception):
    def __init__(self, arg):
        self.arg = arg
        super().__init__(self.arg)

def compute(x):
    result= sqrt(x) + log(2-x) + 1/x
    if result > 1:
        raise MyException('Too big')
    return result

while True:
    x = input('Give a value: ')
    try:
        y = compute(int(x))
    except (ValueError, ZeroDivisionError, MyException) as e:
        print(e)
    else:
        print('Result: ', y)
```

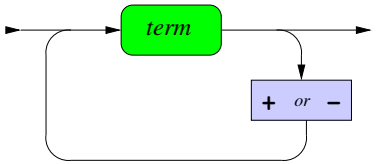
UPPSALA UNIVERSITET

Syntax

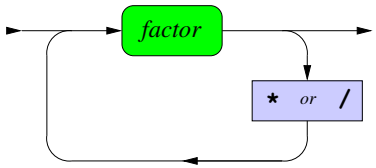
assignment



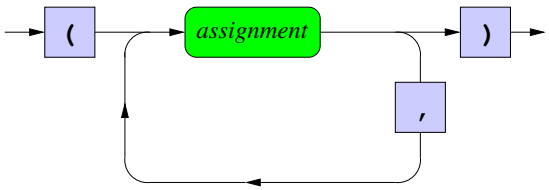
expression



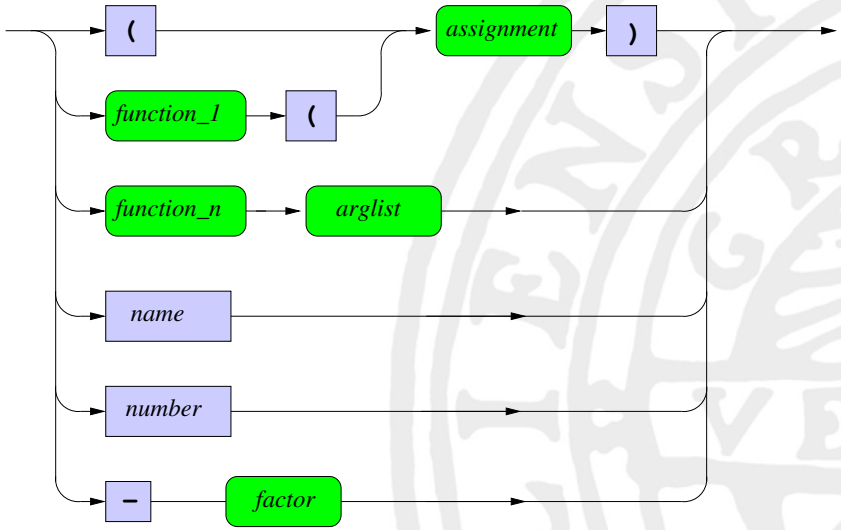
term



arglist



factor



UPPSALA UNIVERSITET

The end

I will record another lecture where I discuss

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

The end

UPPSALA UNIVERSITET

Mer om kalkylatorn

Tom Smedsaas

2022-03-10

- Hela syntaxen.
- Syntaxfel.
- Hur kommer man igång?

UPPSALA UNIVERSITET

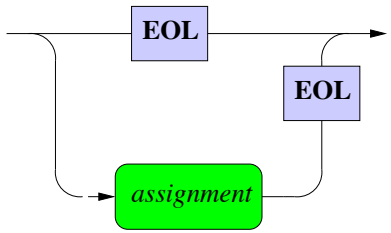
Demonstration

```
Input : 4  - (5-2*2)/(1+1) - (3-1)
Result: 1.5
Input : 1+2+3+4 = x
Result: 10.0
Input : (2*x=y=z) - x/2 - x
Result: 5.0
```

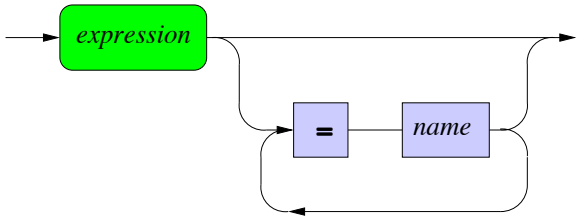
UPPSALA UNIVERSITET

Syntaxdiagram

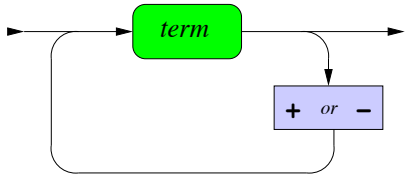
statement



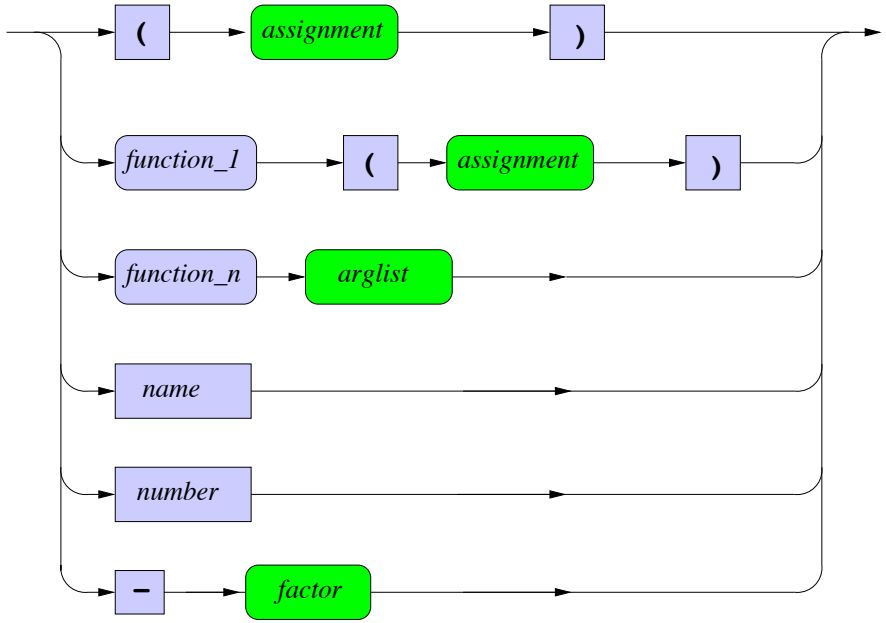
assignment



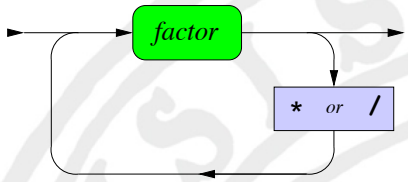
expression



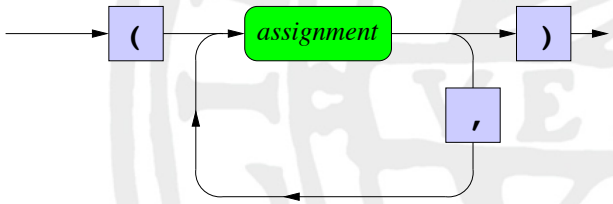
factor



term

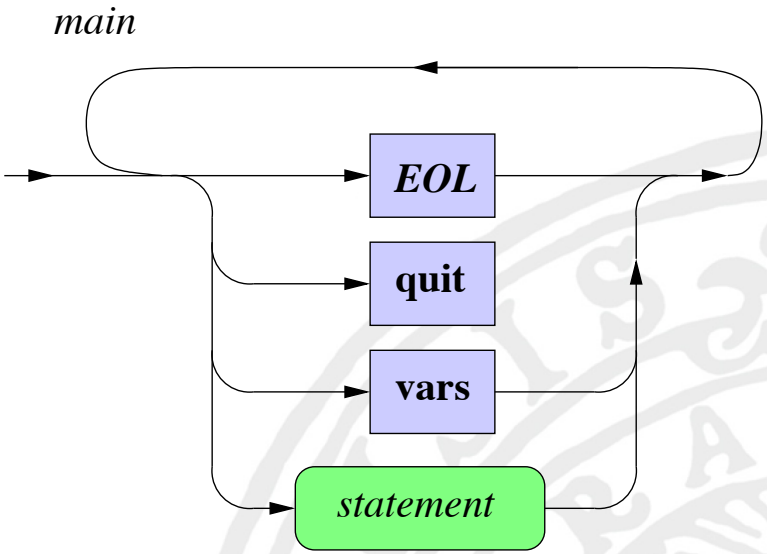


arglist



En `main`-funktion

Vi behöver också en funktion som driver det hela:



Denna funktion börjar med att läsa in rader en från en fil som är bifogad. Tanken är att ni ska fylla på filen med egna tester.

UPPSALA UNIVERSITET

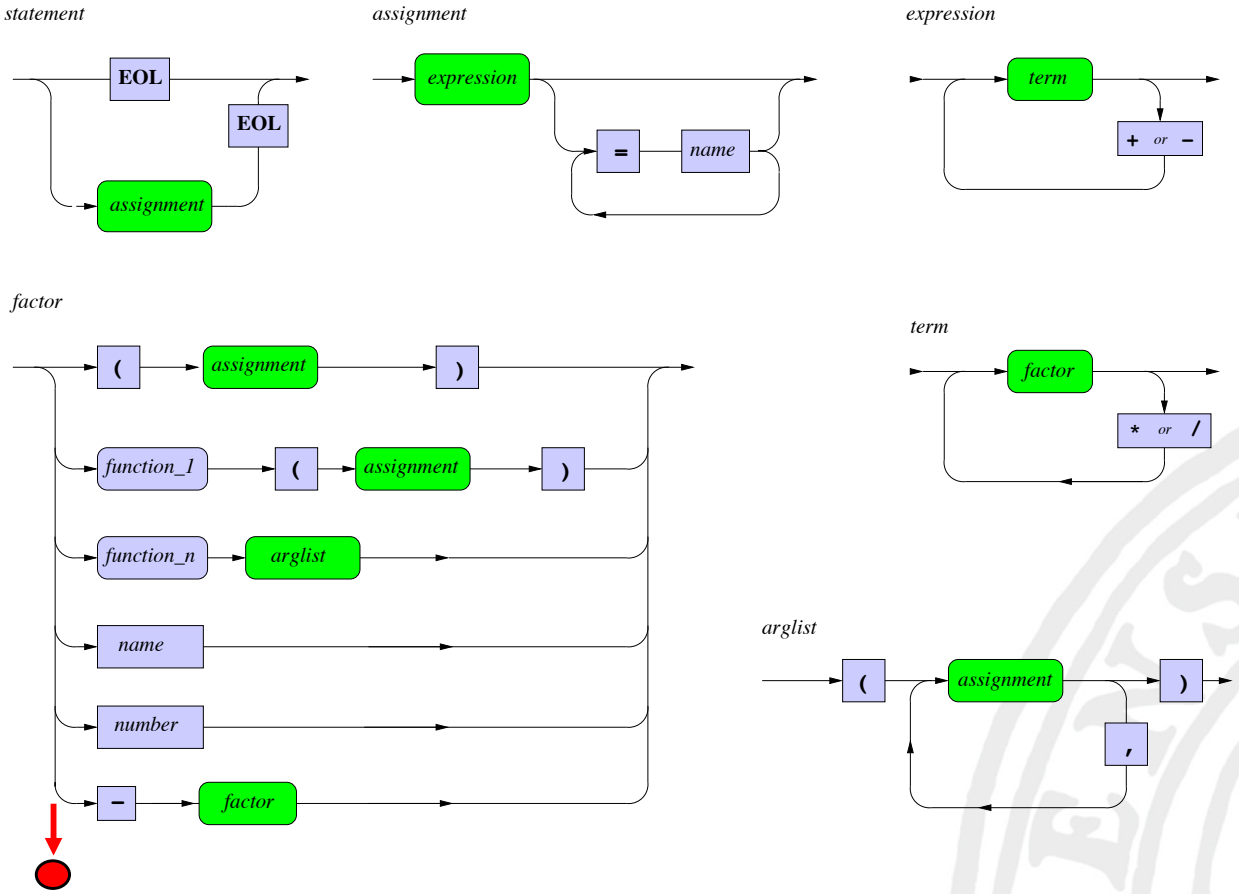
Syntaxfel i indata

Vad händer om användaren skriver ett uttryck som inte är syntaktiskt korrekt?

Exempel:

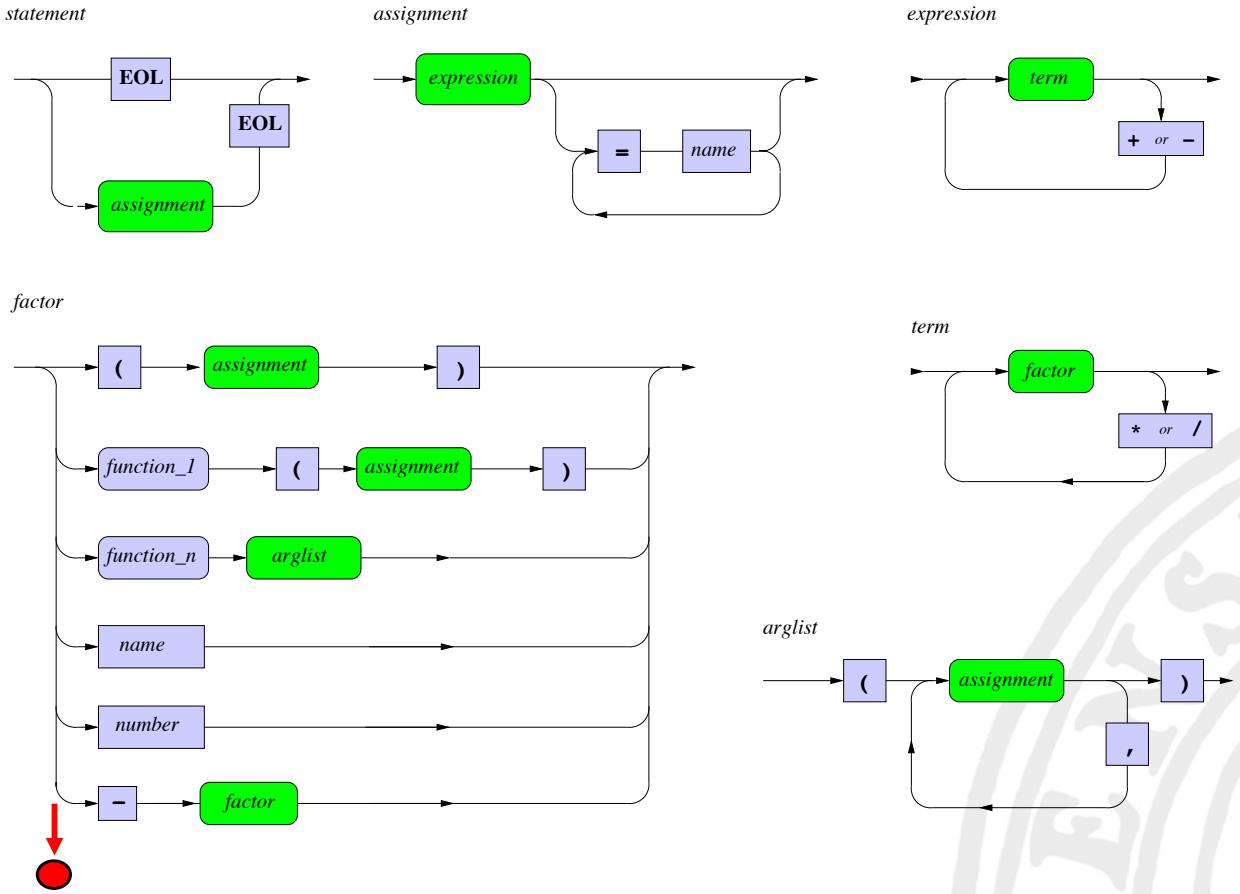
- `1 + * y`
- `+2 + 3`
- `4 = 5`
- `(1=x+1)`
- `1 ) 2`
- `(2`

Input: 1 + * y



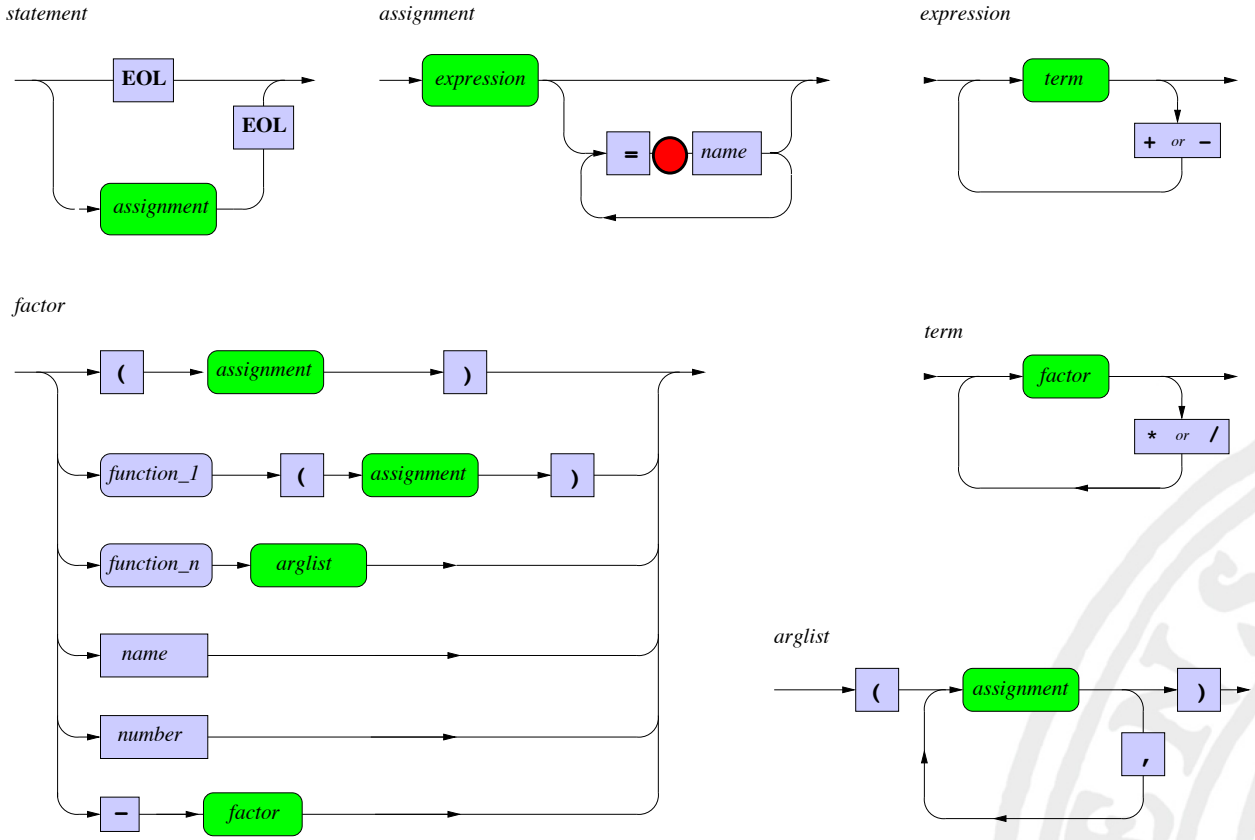
Input : 1 + *y
*** Syntax error: Expected number, word or '('
Error occurred at '*' just after '+'

Input: +2 + 3



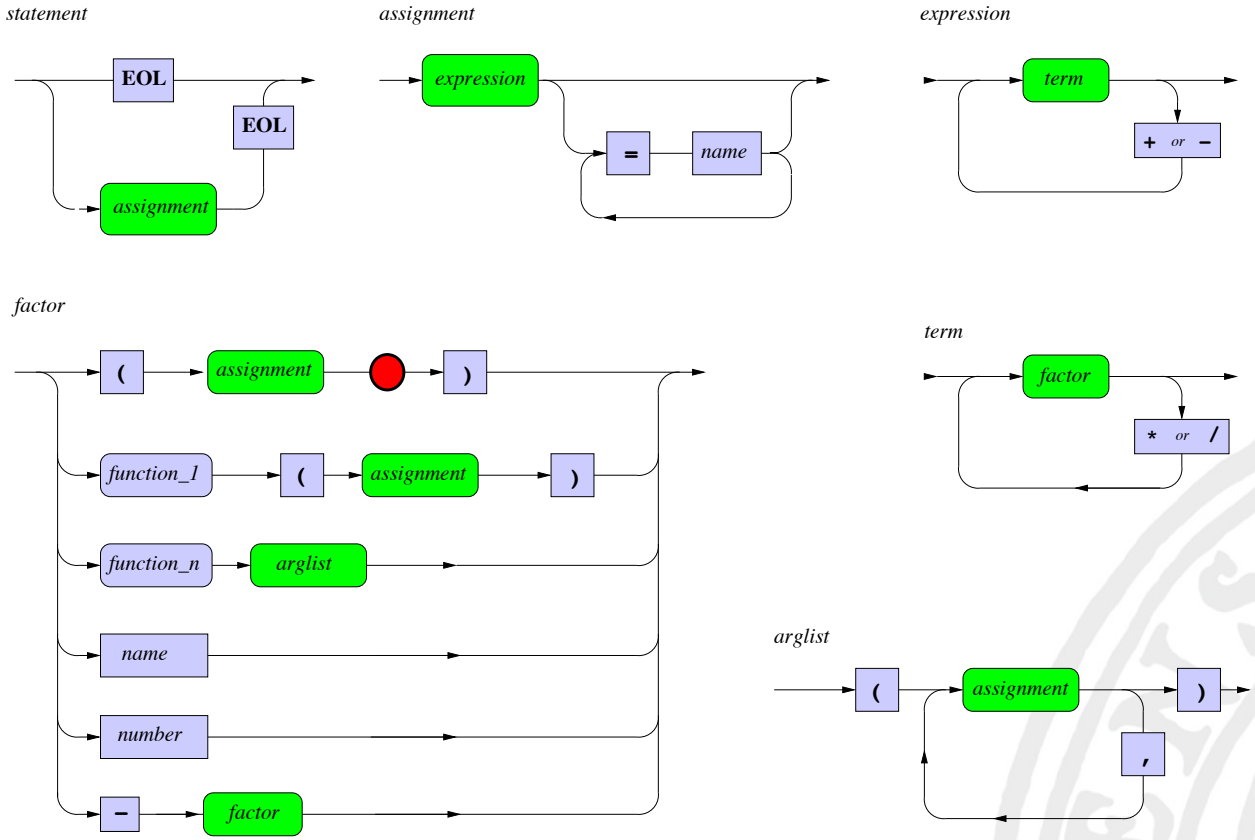
Input : +2+3
*** Syntax error: Expected number, word or '('
Error occurred at '+' just after 'START'

Input: 4 = 5



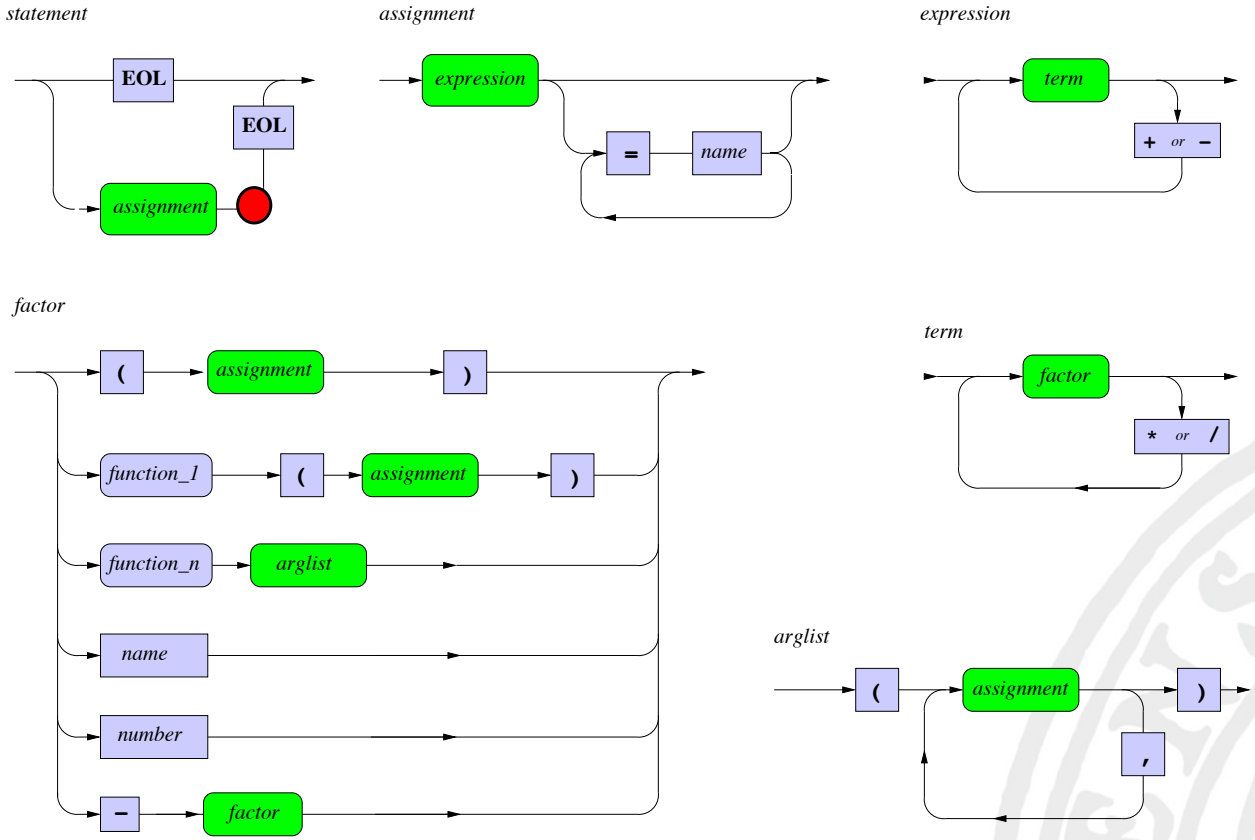
Input : 4 = 5
*** Syntax error: Expected name after '='
Error occurred at ’5' just after '='

Input: (1 = x + 1)



Input : (1 = x + 1)
*** Syntax error: Expected ')'
Error occurred at '+' just after 'x'

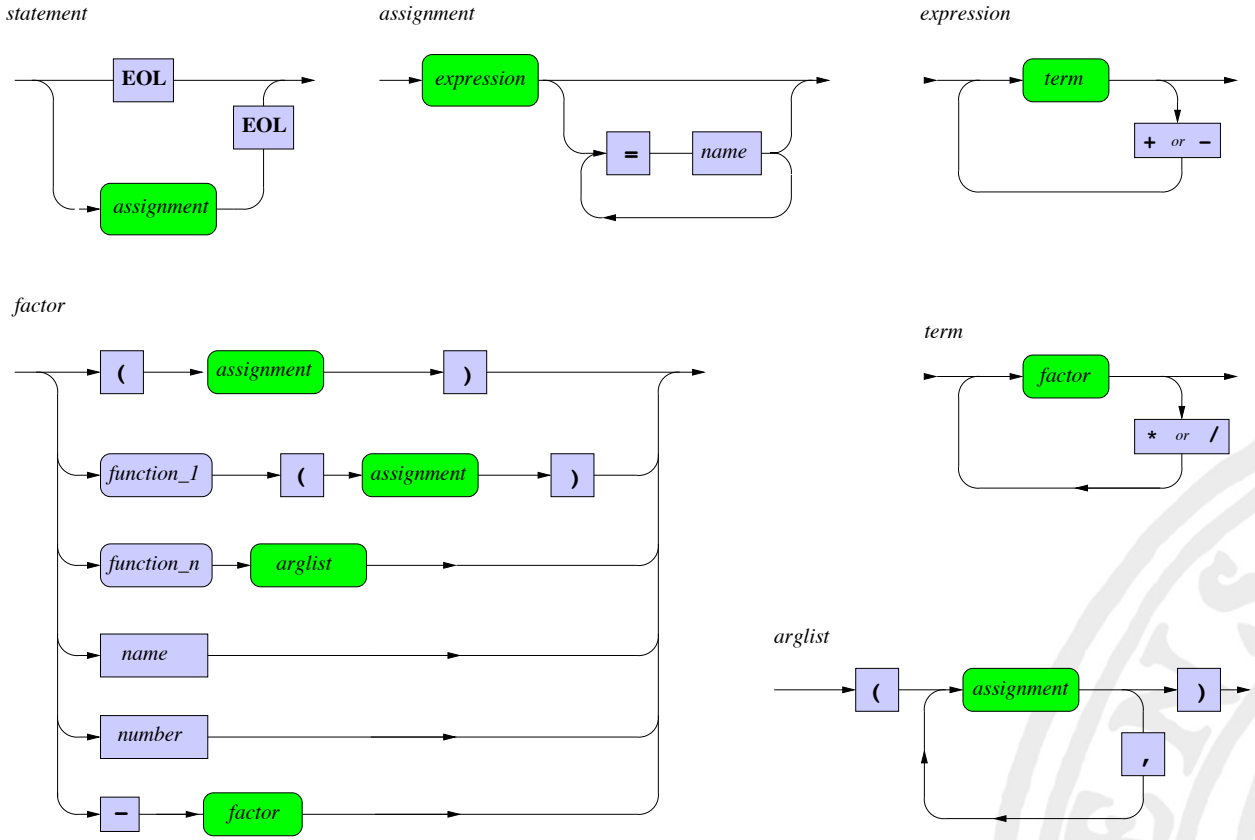
Input: 1) 2



Input : 1) 2
*** Syntax error: Expected end of line
Error occurred at ')' just after '1'

UPPSALA UNIVERSITET

Input: (2

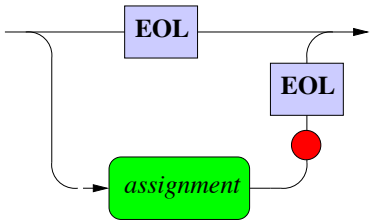


Input : (2
*** Syntax error: Unbalanced parentheses

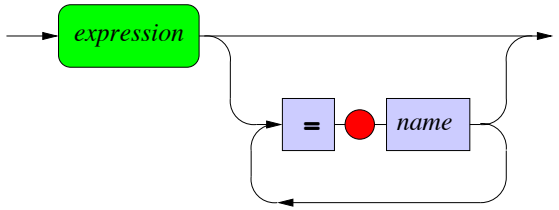
UPPSALA UNIVERSITET

Ställen att kontrollera

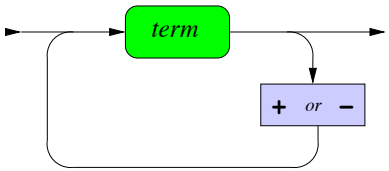
statement



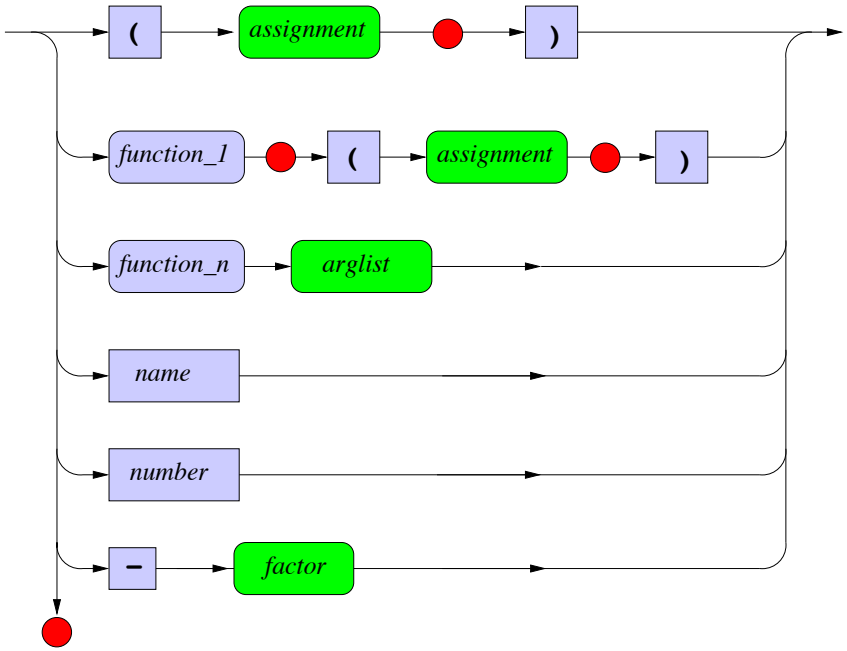
assignment



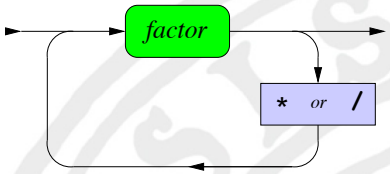
expression



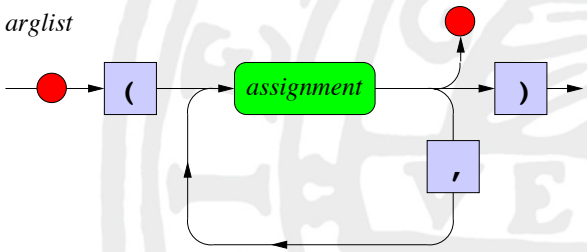
factor



term



arglist



UPPSALA UNIVERSITET

Hur kommer man igång?

- Kör det nedladdade programmet! Det ska fungera direkt för addition, multiplikation och parenteser.
- Lägg till subtraktion och division. Se till att att uttryck som `1+2-3+4` fungerar!
- Lägg in unärt minus dvs så att t ex `-(3+2)` fungerar.
- Skriv klassen `EvaluationError.` Se till att den används vid division med 0. Se till att dessa undantag fångas i *main*.
- Lägg in variabelhanteringen dvs tilldelningen i *assignment* och uppslagning av värden i *factor*. Användning av en odefinierad variabel ska ge ett `EvaluationError.`
- Ha *tydliga*, *korrekta* felmeddelanden om vad som väntades.
- Gå *inte* vidare innan du fått dessa saker att fungera!

UPPSALA UNIVERSITET

The end

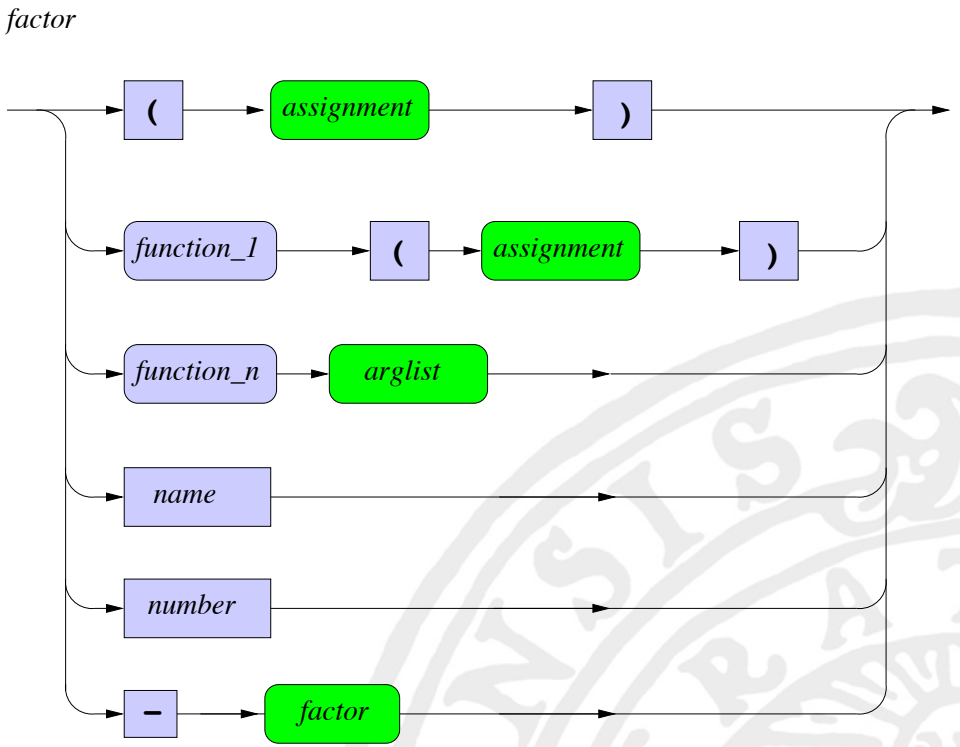
UPPSALA UNIVERSITET

Mer om kalkylatorn

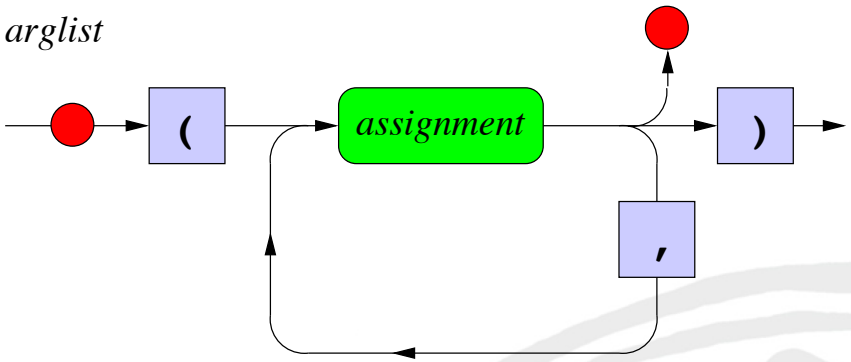
Implementation av funktioner

Funktioner med *ett* argument

```
Input : exp(2)
Result: 7.38905609893065
Input : exp(log(1+1)=y)
Result: 2.0
Input : 3*exp(y) - cos(2*PI)
Result: 5.0
Input : fib(6)
Result: 8
Input : fac(2+3)
Result: 120
Input : sin 2
*** Syntax error:  Expected '(' after function name
Error occurred at '2' just after 'sin'
```



Funktioner med *flera* argument




```
Input : max(exp(0.8), 2*log(1.5))
Result: 2.225540928492468
Input : mean(1,2,3,4,5)
Result: 3.0
Input : min()
*** Syntax error:  Expected number, word or '('
Error occurred at ')' just after ’(’
Input : max 2,3,4
*** Syntax error:  Expected '(' after function name
Error occurred at '2' just after 'max'
```

UPPSALA UNIVERSITET

Evalueringsfel i funktioner

- Illegalt *värde* på argumentet. Exempel: log(2*10-30).
- Illegal *typ* på argumentet. Exempel: fib(3.5).

Sådana saker måste kontrolleras av funktionen själv.

Ni får naturligtvis använda Pythons funktioner men det kan vara bra att skriva en egen `log` som kontrollerar argumentet och sedan använder `math.log`.

UPPSALA UNIVERSITET

The end