

---

# **Conversion Calculator**

**Expressions, types, coercions, names, laziness, functions.**

CS164: Introduction to Programming Languages and Compilers

**Lectures 4 and 5, Fall 2009**

**Instructor: Ras Bodik**

**GSI: Joel Galenson**

**UC Berkeley**

# Administrativia

---

New Section Times:

Wed 12-1 (unchanged time and location)

Wed 5-6 moves to Thu 2-3.

First homework assigned today. Due in a week.

First project due Thu.

Send questions on slides for Lectures 2 and 3 by Thu.

These will be answered as I revise the slides.

*Turn off your cell phones and close laptops.*

# Example programs in our language

---

## Example 1:

**half a dozen pints \* (110 Calories per 12 fl oz) / 25 W in days**

--> 1.704 days

volume \* (energy / volume) / power = time

## Example 2:

**34 knots in mph # speed of S.F. ferry boat**

--> 39.126 mph

# What do we want from the languages

---

- Evaluate arithmetic expressions
  - ... including those with physical units
  - check if conversion is legal (area to volume is not)
  - convert units
- 
- allow the user to extend the language with units
  - add variables
  - and other useful and intriguing stuff

# 1) Language of arithmetic expressions

---

The language is defined as

Syntax: set of valid program strings

Meaning: what the program is to compute.

Syntax: the set of valid programs is infinitely large.

So we define it recursively:

$$\begin{aligned} E &::= n \mid E \text{ op } E \mid ( E ) \\ \text{op} &::= + \mid - \mid * \mid / \mid ^ \end{aligned}$$

E usually denotes set of expressions. n is an integer or a float constant

Examples: 1, 2, 3, 1+2, 1+3, (1+3)\*2, ...

Yes, we could add unary operators:  
 $E ::= \dots \mid \text{unop}$   
 $\text{unop} ::= \text{sqrt} \mid -$

# Meaning of programs

---

Syntax defines what our programs look like:

1, 0.01, 0.12131, 2, 3, 1+2, 1+3, (1+3)\*2, ...

But what do they mean?:

$e_1 + e_2$ : addition over machine integers

but what range of integers?

we borrow Python's implementation of unlimited range integers

... or over double precision floats

if the  $e_1$  or  $e_2$  is float

similar for other ops

# How to represent a program in this language?

---

## concrete syntax

(input program)

1+2

(3+4)\*2

## abstract syntax

(internal program representation)

(‘+’, 1, 2)

(‘\*’, (‘+’, 3, 4), 2)

AST : Abstract Syntax Tree

# The interpreter

---

Recursive descent over the abstract syntax tree

```
def eval(e):
    if type(e) == type(1): return e
    if type(e) == type(1.1): return e
    if type(e) == type(()):
        if e[0] == '+': return eval(e[1]) + eval(e[2])
        if e[0] == '-': return eval(e[1]) - eval(e[2])
        if e[0] == '*': return eval(e[1]) * eval(e[2])
        if e[0] == '/': return eval(e[1]) / eval(e[2])
        if e[0] == '^': return eval(e[1]) ** eval(e[2])
ast = ('*', ('+', 3, 4), 5)
print(eval(ast))
```

# Find the source code at bitbucket

---

The evolution of the interpreter that we implemented in the lecture is at [bitbucket.org](http://bitbucket.org/bodik/cs164fa09/src/bff39887de71/L3-ConversionCalculator/Prep-for-lecture/ConversionCalculator.py):

The interpreter for arithmetic expressions:

<http://bitbucket.org/bodik/cs164fa09/src/bff39887de71/L3-ConversionCalculator/Prep-for-lecture/ConversionCalculator.py>

# Coercions

---

What to do when adding int value and float value?

- 1) coerce int to float, add two floats, or
- 2) coerce float to int, add two ints.

We define the coercion semantics of our language by the rules of the implementation language (Python).

## 2) Add values that are physical units (SI only)

---

Example:

$$(2 \text{ m})^2 \rightarrow 4 \text{ m}^2$$

Concrete syntax:

$E ::= n \mid U \mid E \text{ op } E \mid (E)$

$U ::= m \mid s \mid kg \quad \# \text{ SI units only in Step 2}$

Abstract syntax: represent SI units as string constants

`3 m2`      `(* , 3, (^ , 'm' , 2))`

# Value representation

---

How to represent the result of  $(\text{'^'}, \text{'m'}, 2)$ ?

A pair (numeric value, Unit)

Unit is a map from SI unit to exponent.

```
(\text{'^'}, \text{'m'}, 2)           --> (1, \{ \text{'m'} : 2 \})  
(\text{'*'}, 3, (\text{'^'}, \text{'m'}, 2)) --> (3, \{ \text{'m'} : 2 \})
```

# The interpreter code

---

```
def eval(e):
    if type(e) == type(1): return (e,{})
    if type(e) == type('m'): return (1,{e:1})
    if type(e) == type(()):
        if e[0] == '+': return add(eval(e[1]), eval(e[2]))
        ...
def sub((n1,u1), (n2,u2)):
    if u1 != u2: raise Exception("Adding incompatible units")
    return (n1-n2,u1)
def mul((n1,u1), (n2,u2)):
    return (n1*n2,mulUnits(u1,u2))
```

Read rest of code at:

<http://bitbucket.org/bodik/cs164fa09/src/9d975a5e8743/L3-ConversionCalculator/Prep-for-lecture/ConversionCalculator.py>

## Step 3: add non-SI units

---

Trivial extension to syntax

$E ::= n \mid U \mid E \text{ op } E \mid (E)$

$U ::= m \mid s \mid kg \mid \mathbf{ft} \mid \mathbf{year} \mid \dots$

How to extend the interpreter?

All we need is convert **ft** to **m** at the leaves of the AST

# The code

---

```
def eval(e):
    if type(e) == type(1): return (e,{})
    if type(e) == type(1.1): return (e,{})
    ...
    
def lookupUnit(u):
    return { 'm' : (1, {'m':1}),
             'ft' : (0.3048, {'m':1}),
             'in' : (0.0254, {'m':1}),
             's' : (1, {'s':1}),
             'year': (31556926, {'s':1}),
             'kg' : (1, {'kg':1}),
             'lb' : (0.45359237, {'kg':1})}
    }[u];
```

Rest of code at :

<http://bitbucket.org/bodik/cs164fa09/src/c73c51cfce36/L3-ConversionCalculator/Prep-for-lecture/ConversionCalculator.py>

# Coercion revisited

---

To what should "1 m / year" evaluate?

our interpreter outputs 0 m / s

The value  $1 / 31556926 * \text{m} / \text{s}$  was rounded to zero

Python coercion rules not suitable for the calculator.

We need our own.

Keep value in integer type whenever possible.

Convert to float when precision would be lost.

Read the code here:

<http://bitbucket.org/bodik/cs164fa09/src/204441df23c1/L3-ConversionCalculator/Prep-for-lecture/ConversionCalculator.py>

## Step 4: Add conversion

---

Example:

3 ft/s **in** m/year  $\rightarrow 28\ 855\ 653.1$  m / year

Language of Step 3:

$E ::= n \mid U \mid E \text{ op } E \mid (E)$

$U ::= m \mid s \mid kg \mid J \mid ft \mid in \mid \dots$

$\text{op} ::= + \mid - \mid * \mid \varepsilon \mid / \mid ^$

Let's extend this language for Step 4.

We need to add the construct "**E in C**"

# Where in the program can "E in C" appear?

---

## Attempt 1:

$E ::= n \mid U \mid E \text{ op } E \mid (E) \mid E \text{ in } C$

That is, is the construct "E in C" a kind of expression?

If yes, we must allow it wherever expressions appear.

For example  $\text{in} (2 \text{ m in ft}) + 3 \text{ km}$ .

For that, E in C must produce a value.

## Attempt 2:

$S ::= E \mid E \text{ in } C$

$E ::= n \mid U \mid E \text{ op } E \mid (E)$

"E in C" is a top-level statement.

It influences how the value of E is printed.

# What are the valid forms of C?

---

## Attempt 1:

$C ::= U \text{ op } U$

$U ::= m | s | kg | ft | J | \dots$

$\text{op} ::= + | - | * | \varepsilon | / | ^$

Examples valid programs:

Examples of invalid programs:

## Attempt 2:

$C ::= U * U | U U | U / U | U ^ n$

$U ::= m | s | kg | ft | J | \dots$

# Evaluation of C

---

What values do we need to obtain from C?

1. conversion ratio between the unit C and the corresponding SI unit

ex:  $(\text{ft/year}) / (\text{m/s}) = 9.65873546 \times 10^{-9}$

1. a representation of C to print out

ex:  $\text{ft} * \text{m} * \text{ft} \rightarrow \{\text{ft}:2, \text{m}:1\}$

# The code

---

The interpreter:

<http://bitbucket.org/bodik/cs164fa09/src/fd06b6dfob9c/L3-ConversionCalculator/Prep-for-lecture/ConversionCalculator.py>

The parser (FYI):

[http://bitbucket.org/bodik/cs164fa09/src/fd06b6dfob9c/L3-ConversionCalculator/Prep-for-lecture/Lo\\_parser.py](http://bitbucket.org/bodik/cs164fa09/src/fd06b6dfob9c/L3-ConversionCalculator/Prep-for-lecture/Lo_parser.py)