

Calculator

Announcements

Exceptions

Raise Statements

Python exceptions are raised with a raise statement

```
raise <expression>
```

<expression> must evaluate to a subclass of BaseException or an instance of one

Exceptions are constructed like any other object. E.g., `TypeError('Bad argument!')`

`TypeError` -- A function was passed the wrong number/type of argument

`NameError` -- A name wasn't found

`KeyError` -- A key wasn't found in a dictionary

`RecursionError` -- Too many recursive calls

(Demo)

Try Statements

Try statements handle exceptions

```
try:
    <try suite>
except <exception class> as <name>:
    <except suite>
...
```

Execution rule:

The <try suite> is executed first

If, during the course of executing the <try suite>, an exception is raised that is not handled otherwise, and

If the class of the exception inherits from <exception class>, then

The <except suite> is executed, with <name> bound to the exception

5

Handling Exceptions

Exception handling can prevent a program from terminating

```
>>> try:
    x = 1/0
    except ZeroDivisionError as e:
        print('handling a', type(e))
    x = 0

handling a <class 'ZeroDivisionError'>
>>> x
0
```

(Demo)

6

Example: Reduce

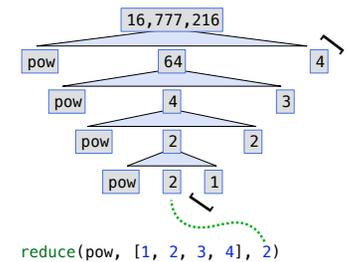
Reducing a Sequence to a Value

```
def reduce(f, s, initial):
    """Combine elements of s pairwise using f, starting with initial.
```

E.g., `reduce(mul, [2, 4, 8], 1)` is equivalent to `mul(mul(mul(1, 2), 4), 8)`.

```
>>> reduce(mul, [2, 4, 8], 1)
64
.....
```

`f` is ...
a two-argument function
`s` is ...
a sequence of values that can be the second argument
`initial` is ...
a value that can be the first argument



(Demo)

8

Programming Languages

Programming Languages

A computer typically executes programs written in many different programming languages

Machine languages: statements are interpreted by the hardware itself

- A fixed set of instructions invoke operations implemented by the circuitry of the central processing unit (CPU)
- Operations refer to specific hardware memory addresses; no abstraction mechanisms

High-level languages: statements & expressions are interpreted by another program or compiled (translated) into another language

- Provide means of abstraction such as naming, function definition, and objects
- Abstract away system details to be independent of hardware and operating system

Python 3

```
def square(x):  
    return x * x
```

from dis import dis
dis(square)

Python 3 Byte Code

```
LOAD_FAST      0 (x)  
LOAD_FAST      0 (x)  
BINARY_MULTIPLY  
RETURN_VALUE
```

Metalinguistic Abstraction

A powerful form of abstraction is to define a new language that is tailored to a particular type of application or problem domain

Type of application: Erlang was designed for concurrent programs. It has built-in elements for expressing concurrent communication. It is used, for example, to implement chat servers with many simultaneous connections

Problem domain: The MediaWiki mark-up language was designed for generating static web pages. It has built-in elements for text formatting and cross-page linking. It is used, for example, to create Wikipedia pages

A programming language has:

- **Syntax:** The legal statements and expressions in the language
- **Semantics:** The execution/evaluation rule for those statements and expressions

To create a new programming language, you either need a:

- **Specification:** A document describe the precise syntax and semantics of the language
- **Canonical Implementation:** An interpreter or compiler for the language

Parsing

Reading Scheme Lists

A Scheme list is written as elements in parentheses:

(<element_0> <element_1> ... <element_n> A Scheme list

Each <element> can be a combination or primitive

(+ (* 3 (+ (* 2 4) (+ 3 5))) (+ (- 10 7) 6))

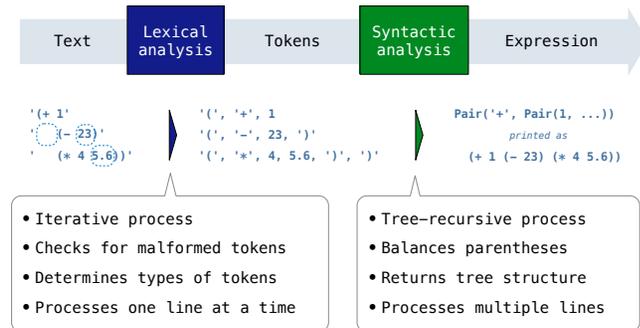
The task of parsing a language involves coercing a string representation of an expression to the expression itself

(Demo)

13

Parsing

A Parser takes text and returns an expression



14

Syntactic Analysis

Syntactic analysis identifies the hierarchical structure of an expression, which may be nested

Each call to `scheme_read` consumes the input tokens for exactly one expression

```
'(', '+', 1, '(', '-', 23, ')', '(', '*', 4, 5.6, ')', ')'
```

Base case: symbols and numbers

Recursive call: `scheme_read` sub-expressions and combine them

15

Scheme-Syntax Calculator

(Demo)

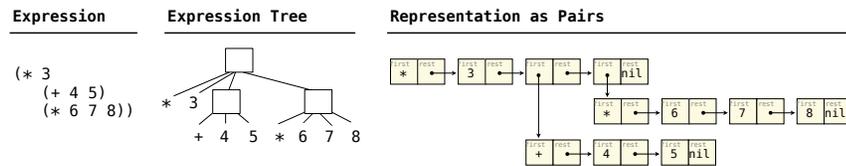
Calculator Syntax

The Calculator language has primitive expressions and call expressions. (That's it!)

A primitive expression is a number: 2 -4 5.6

A call expression is a combination that begins with an operator (+, -, *, /) followed by 0 or more expressions: (+ 1 2 3) (/ 3 (+ 4 5))

Expressions are represented as Scheme lists (Pair instances) that encode tree structures.



Calculator Semantics

The value of a calculator expression is defined recursively.

Primitive: A number evaluates to itself.

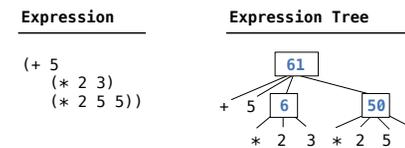
Call: A call expression evaluates to its argument values combined by an operator.

+: Sum of the arguments

*: Product of the arguments

-: If one argument, negate it. If more than one, subtract the rest from the first.

/: If one argument, invert it. If more than one, divide the rest from the first.



Evaluation

The Eval Function

The eval function computes the value of an expression, which is always a number

It is a generic function that dispatches on the type of the expression (primitive or call)

Implementation

```
def calc_eval(exp):
    if isinstance(exp, (int, float)):
        return exp
    elif isinstance(exp, Pair):
        arguments = exp.rest.map(calc_eval)
        return calc_apply(exp.first, arguments)
    else:
        raise TypeError
```

Recursive call returns a number for each operand

A Scheme list of numbers

Language Semantics

A number evaluates...

to itself

A call expression evaluates...

to its argument values combined by an operator

Handling Exceptions

An interactive interpreter prints information about each error

A well-designed interactive interpreter should not halt completely on an error, so that the user has an opportunity to try again in the current environment

(Demo)