# Why SageX is not quite Pyrex

Robert Bradshaw

June 15, 2007

# What is Pyrex?

*Pyrex lets you write code that mixes Python and C data types any way you want, and compiles it into a C extension for Python.*
— Greg Ewing (Author)

# What is Pyrex?

- Pseudo-Python to C compiler
- Language extensions for statically declaring types
  - Potentially massive speedups
  - Integration with external libraries
- Python memory management and Python object $\leftrightarrow$ c data type coercions done automatically.

# Pyrex and SAGE

- Compiled language necessary for a serious CAS
- Pyrex provides a consistent interface

# Pyrex and SAGE

- Compiled language necessary for a serious CAS
- Pyrex provides a consistent interface
- Pyrex provides an easy migration path
  - Python code easily becomes Pyrex code (even incrementally)
  - Python developers easily become Pyrex developers

# Pyrex and SAGE

- Compiled language necessary for a serious CAS
- Pyrex provides a consistent interface
- Pyrex provides an easy migration path
  - Python code easily becomes Pyrex code (even incrementally)
  - Python developers easily become Pyrex developers
- Pyrex provides direct and natural access to both the Python and C environments.

# Why SageX?

Up until summer 2006, all Pyrex code lived in `sage/ext`

# Why SageX?

Up until summer 2006, all Pyrex code lived in `sage/ext`

- Not modular

# Why SageX?

Up until summer 2006, all Pyrex code lived in `sage/ext`

- Not modular
- Not maintainable

## Why SageX?

Up until summer 2006, all Pyrex code lived in `sage/ext`

- Not modular
- Not maintainable

This was due to the inability of Pyrex to do cross-directory imports.

# sage/ext

June 15, 2006

cdefs.pxi
coerce.pyx
congroup_pyx.pyx
dense_matrix_pyx.pxd
dense_matrix_pyx.pyx
element.pxd
element.pyx
gens.pxd
gens.pyx
gens_py.py
gmp.pxi
gmpy.h
group.pxd
group.pyx
heilbronn.pyx
integer.pxd
integer.pxi
integer.pyx
interrupt.c
interrupt.h
interrupt.pxi
intmod_pyx.pyx
module.pxd
module.pyx
mpc.pyx
mpfr.pyx
mpn_pylong.c
mpn_pylong.h
mpz_pylong.c
mpz_pylong.h
p1list.pxd
p1list.pyx
polynomial_pyx.pxi
polynomial_pyx.pyx
pymemcompat.h
rational.pxd
rational.pxi
rational.pyx
ring.pxd
ring.pyx
sage_object.pxd
sage_object.pyx
search.pyx
sparse_matrix_pyx.pyx
sparse_poly.pxi
sparse_poly.pyx

# sage/ext

## Now?

```
rational.pyx
arith.pxd
arith.pxi
arith.pyx
arith_gmp.pxd
arith_gmp.pxi
arith_gmp.pyx
bernoulli_mod_p.pyx
binary_search.pxi
callback.pyx
cdefs.pxi
coerce.pxd
coerce.pxi
coerce.pyx
complex_double.pxd
complex_double.pyx
complex_double_vector.pxd
complex_double_vector.pyx
complex_number.pxd
complex_number.pyx
congroup_pyx.c
congroup_pyx.pyx
decl.pxi
dwt.pxd
dwt.pyx
ec.pyx
element.pxd
element.pyx
expnums.pyx
fft.pxd
fft.pyx
finite_field_givaro.pxd
finite_field_givaro.pyx
free_module_element.pxd
free_module_element.pyx
frobenius.pyx
gen.pxd
gen.pxi
gen.pyx
gmp.pxi
graph_fast.pyx
group.pxd
group.pyx
gsl.pxi
gsl_airy.pxi
gsl_array.pxd
gsl_array.pyx
gsl_bessel.pxi
gsl_blas.pxi
gsl_blas_types.pxi
gsl_block.pxi
gsl_chebyshev.pxi
gsl_clausen.pxi
gsl_combination.pxi
gsl_complex.pxi
gsl_coulomb.pxi
gsl_coupling.pxi
gsl_dawson.pxi
gsl_debye.pxi
gsl_diff.pxi
gsl_dilog.pxi
gsl_eigen.pxi
gsl_elementary.pxi
gsl_ellint.pxi
gsl_elljac.pxi
gsl_erf.pxi
gsl_errno.pxi
gsl_exp.pxi
gsl_expint.pxi
gsl_fermi_dirac.pxi
gsl_fft.pxi
gsl_fit.pxi
gsl_gamma.pxi
gsl_gegenbauer.pxi
gsl_histogram.pxi
gsl_hyperg.pxi
gsl_integration.pxi
gsl_interp.pxi
gsl_laguerre.pxi
gsl_lambert.pxi
gsl_legendre.pxi
gsl_linalg.pxi
gsl_log.pxi
gsl_math.pxi
gsl_matrix.pxi
gsl_matrix_complex.pxi
gsl_min.pxi
gsl_mode.pxi
gsl_monte.pxi
gsl_ntuple.pxi
gsl_odeiv.pxi
gsl_permutation.pxi
gsl_poly.pxi
gsl_pow_int.pxi
gsl_psi.pxi
gsl_qrng.pxi
gsl_random.pxi
gsl_rng.pxi
gsl_roots.pxi
gsl_sf_result.pxi
gsl_sort.pxi
gsl_statistics.pxi
gsl_sum.pxi
gsl_synchrotron.pxi
gsl_transport.pxi
gsl_trig.pxi
gsl_vector.pxi
gsl_vector_complex.pxi
gsl_wavelet.pxi
gsl_zeta.pxi
gslonly.pxi
hanke.pyx
heilbronn.pyx
init.pxi
integer.pxd
integer.pxi
integer.pyx
integer_mod.pxd
integer_mod.pyx
integer_ring.pxd
integer_ring.pyx
integration.pyx
interactive_constructors_c.pyx
interpolation.pxd
interpolation.pyx
interrupt.pxi
laurent_series_ring_element.pxd
laurent_series_ring_element.pyx
linbox.pxd
linbox.pyx
local_generic_element.pxd
local_generic_element.pyx
math.pxi
matrix.pxd
matrix.pyx
matrix0.pxd
matrix0.pyx
matrix1.pxd
matrix1.pyx
matrix2.pxd
matrix2.pyx
matrix_RR_dense.pyx
matrix_complex_double_dense.pxd
matrix_complex_double_dense.pyx
matrix_cyclo_dense.pxd
matrix_cyclo_dense.pyx
matrix_cyclo_sparse.pxd
matrix_cyclo_sparse.pyx
matrix_dense.pxd
matrix_dense.pyx
matrix_domain_dense.pxd
matrix_domain_dense.pyx
matrix_domain_sparse.pxd
matrix_domain_sparse.pyx
matrix_field_dense.pyx
matrix_field_sparse.pxd
matrix_field_sparse.pyx
matrix_generic_dense.pxd
matrix_generic_dense.pyx
matrix_generic_sparse.pxd
matrix_generic_sparse.pyx
matrix_integer_2x2.pxd
matrix_integer_2x2.pyx
matrix_integer_dense.pxd
matrix_integer_dense.pyx
matrix_integer_sparse.pxd
matrix_integer_sparse.pyx
matrix_mod2_dense.pxd
matrix_mod2_dense.pyx
matrix_modn_dense.pxd
matrix_modn_dense.pyx
matrix_modn_sparse.pxd
matrix_modn_sparse.pyx
matrix_padic_capped_relative_dense.pxd
matrix_padic_capped_relative_dense.pyx
matrix_rational_dense.pxd
matrix_rational_dense.pyx
matrix_rational_sparse.pxd
matrix_rational_sparse.pyx
matrix_real_double_dense.pxd
matrix_real_double_dense.pyx
matrix_sparse.pxd
matrix_sparse.pyx
matrix_window.pxd
matrix_window.pyx
matrix_window_modn_dense.pxd
matrix_window_modn_dense.pyx
memory.pyx
misc.pxi
misc.pyx
module.pxd
module.pyx
mpc.pyx
mpfi.pxi
mpfr.pxi
multi_modular.pxd
multi_modular.pyx
multi_polynomial.pxd
multi_polynomial.pyx
multi_polynomial_libsingular.pxd
multi_polynomial_libsingular.pyx
multi_polynomial_ring_generic.pxd
multi_polynomial_ring_generic.pyx
mutability.pxd
mutability.pyx
mwrank.pyx
ntl.pxd
ntl.pxi
ntl.pyx
number_field_element.pxd
number_field_element.pyx
ode.pxd
ode.pyx
p1list.pxd
p1list.pyx
padic_base_generic_element.pxd
padic_base_generic_element.pyx
padic_capped_absolute_element.pxd
padic_capped_absolute_element.pyx
padic_capped_relative_element.pxd
padic_capped_relative_element.pyx
padic_fixed_mod_element.pxd
padic_fixed_mod_element.pyx
padic_generic_element.pxd
padic_generic_element.pxi
padic_generic_element.pyx
parent.pxd
parent.pyx
parent_base.pxd
parent_base.pyx
parent_gens.pxd
parent_gens.pyx
pari_err.pxi
polydict.pyx
polynomial_compiled.pxd
polynomial_compiled.pyx
polynomial_element.pxd
polynomial_element.pyx
polynomial_pyx.pyx
pow_computer.pxd
pow_computer.pyx
power_series_ring_element.pxd
power_series_ring_element.pyx
probability_distribution.pxd
probability_distribution.pyx
pthread.pxi
python.pxi
python_bool.pxi
python_complex.pxi
python_dict.pxi
python_exc.pxi
python_float.pxi
python_function.pxi
python_instance.pxi
python_int.pxi
python_iterator.pxi
python_list.pxi
python_long.pxi
python_mapping.pxi
python_mem.pxi
python_method.pxi
python_module.pxi
python_number.pxi
python_object.pxi
python_parse.pxi
python_ref.pxi
python_sequence.pxi
python_set.pxi
python_string.pxi
python_tuple.pxi
python_type.pxi
random.pxi
rational.pxd
rational.pxi
real_double.pxd
real_double.pxi
real_double.pyx
real_double_vector.pxd
real_double_vector.pyx
real_mpfi.pxd
real_mpfi.pyx
real_mpfr.pxd
real_mpfr.pyx
real_rqdf.pxd
real_rqdf.pyx
reset.pyx
right_cosets.pyx
ring.pxd
ring.pyx
sage_object.pxd
sage_object.pyx
sagex_c.pyx
sagex_ds.pxd
sagex_ds.pyx
search.pyx
setlvalue.pxi
sig.pyx
singular-cdefs.pxi
singular.pxd
singular.pxi
singular.pyx
solve.pyx
sparse_poly.pyx
stdio.pxi
stdsage.pxi
strassen.pyx
template.pxd
template.pyx
test.pxd
test.pyx
to_gen.pxi
var.pyx
vector_integer_dense.pxd
vector_integer_dense.pyx
vector_integer_sparse_c.pxi
vector_integer_sparse_h.pxi
vector_modn_dense.pxd
vector_modn_dense.pyx
vector_modn_sparse_c.pxi
vector_modn_sparse_h.pxi
vector_rational_dense.pxd
vector_rational_dense.pyx
vector_rational_sparse.pyx
vector_rational_sparse_c.pxi
vector_rational_sparse_h.pxi
```

# Why SageX?

William Stein and Martin Albrecht wrote a straightforward patch to fix this issue.

# Why SageX?

William Stein and Martin Albrecht wrote a straightforward patch to fix this issue.

- Patch not accepted upstream as Greg Ewing does not consider this a bug.

# Why SageX?

William Stein and Martin Albrecht wrote a straightforward patch to fix this issue.

- Patch not accepted upstream as Greg Ewing does not consider this a bug.
- Other valuable (to us) patches not accepted. E.g. introspection.

# Why SageX?

William Stein and Martin Albrecht wrote a straightforward patch to fix this issue.

- Patch not accepted upstream as Greg Ewing does not consider this a bug.
- Other valuable (to us) patches not accepted. E.g. introspection.

Eventually we came to the conclusion that we would have to maintain our own branch. Hopefully, someday, they will merge again.

# Enhancements

- List Comprehension
- In-place arithmetic
- Conditional expressions
- Useful `sizeof`
- `inline` modifier for c functions
- Assignment on declaration
- `for ... from ... by ...`
- `bint` (boolean int) type

# Enhancements - List Comprehension

New node type ListComprehensionAppendNode. Use the existing for/if nodes. Modify the parser accordingly. Note that appending to lists is fast (from c) as lists do predictive allocation.

```
L = []
for x in A:
    if x.is_square()
        L.append(2*x) # attribute lookup and method call
```

```
L = [2*x for x in A if x.is_square()]
```

# Enhancements - In-place operation

For example

`x += 1`     or     `L[f(x)+y] += g(x)`

This is a bit tricky because `f(x)+y` must not be evaluated twice as it might have side effects. We must evaluate first but postpone any cleanup of the index until the very end.

## Enhancements - bint type

- In c, `ints` are used for truth values

# Enhancements - bint type

- In c, `int`s are used for truth values
- when coercing into Python, they become python `int`s rather than `True`/`False`, so explicit coercion is needed

# Enhancements - bint type

```
bool(x == y)
```

# Enhancements - bint type

```
__pyx_1 = __Pyx_GetName(__pyx_b, __pyx_n_bool); if (!__pyx_1)
__pyx_filename = __pyx_f[0]; __pyx_lineno = 3; goto __pyx_L1;
__pyx_2 = PyInt_FromLong((__pyx_v_4bool_x ==
__pyx_v_4bool_y)); if (!__pyx_2) __pyx_filename = __pyx_f[0];
__pyx_lineno = 3; goto __pyx_L1;
__pyx_3 = PyTuple_New(1); if (!__pyx_3) __pyx_filename =
__pyx_f[0]; __pyx_lineno = 3; goto __pyx_L1;
PyTuple_SET_ITEM(__pyx_3, 0, __pyx_2);
__pyx_2 = 0;
__pyx_2 = PyObject_CallObject(__pyx_1, __pyx_3); if (!__pyx_2)
__pyx_filename = __pyx_f[0]; __pyx_lineno = 3; goto __pyx_L1;
Py_DECREF(__pyx_1); __pyx_1 = 0;
Py_DECREF(__pyx_3); __pyx_3 = 0;
```

## Enhancements - bint type

- c `int`s often represent truth values
- when coercing into Python, they become python `int`s rather than `True`/`False`, so explicit coercion is needed
- For storing truth values in SageX, use the `bint` type, which is a c `int` but will get coerced to (and from!) Python as a boolean.

# Enhancements - bint type

```
__pyx_1 = __Pyx_PyBool_FromLong((__pyx_v_4bool_x ==
__pyx_v_4bool_y)); if (!__pyx_1) __pyx_filename = __pyx_f[0];
__pyx_lineno = 3; goto __pyx_L1;
```

# Global Optimizations

- Loops
- Indexing
- Builtin methods
- other __builtin__ objects

## Global Optimizations - Loops

- Pyrex implements `for x in A` using an iterator.

# Global Optimizations - Loops

- Pyrex implements `for x in A` using an iterator.
- If `A` is a list, one can loop over its elements as a `PyObject**`

# Global Optimizations - Loops

- Pyrex implements `for x in A` using an iterator.
- If `A` is a list, one can loop over its elements as a `PyObject**`
- `PyList_CheckExact` called *at runtime* to see if one can use the faster method.

# Global Optimizations - Loops

- Pyrex implements `for x in A` using an iterator.
- If `A` is a list, one can loop over its elements as a `PyObject**`
- `PyList_CheckExact` called *at runtime* to see if one can use the faster method.
  - Common enough, and enough of a gain, to be worth the overhead.

# Global Optimizations - Loops

```
sage:  time loop(A, 1000)
CPU time:  0.58 s, Wall time:  0.60 s
sage:  time loop_iter(A, 1000)
CPU time:  1.33 s, Wall time:  1.36 s
```

## Global Optimizations - Indexing

- Pyrex indexes `L[i]` using the __getitem__ method.

# Global Optimizations - Indexing

- Pyrex indexes `L[i]` using the __getitem__ method.
- If `A` is a list or tuple and `i` an int one can access its members directly.

# Global Optimizations - Indexing

- Pyrex indexes `L[i]` using the __getitem__ method.
- If `A` is a list or tuple and `i` an int one can access its members directly.
- Again, check *at runtime* to see if one can use the faster method.

# Global Optimizations - Indexing

```
sage:  time index_c_int(A, 1000)
CPU time:  0.76 s, Wall time:  0.77 s
sage:  time index_py_int(A, 1000)
CPU time:  5.34 s, Wall time:  5.39 s
```

# Global Optimizations - Builtin methods

- Many methods, such as `len`, `hash`, `isinstance` have fast Python/C API equivalents.

# Global Optimizations - Builtin methods

- Many methods, such as `len`, `hash`, `isinstance` have fast Python/C API equivalents.
- All we have to do is populate the original namespace with the Python/C API equivalents.

# Global Optimizations - Builtin methods

- Many methods, such as `len`, `hash`, `isinstance` have fast Python/C API equivalents.
- All we have to do is populate the original namespace with the Python/C API equivalents.
  - cdef extern len "PyObject_Size" (object o)

# Global Optimizations - Builtin methods

- Many methods, such as `len, hash, isinstance` have fast Python/C API equivalents.
- All we have to do is populate the original namespace with the Python/C API equivalents.
  - cdef extern len "PyObject_Size" (object o)
- But `int()`, for example, is not really a function (it is a class) so we'd have to look at the context at least.

# Global Optimizations - Indexing

```
sage: time len_c(A, 10^7)
CPU time: 0.09 s, Wall time: 0.10 s
sage: time len_py(A, 10^7)
CPU time: 0.98 s, Wall time: 0.99 s
```

## Global Optimizations - other __builtin__ objects

- We can still greatly increase the speed of stuff like the global class `int`.

# Global Optimizations - other __builtin__ objects

- We can still greatly increase the speed of stuff like the global class `int`.
- Pyrex performs a module lookup on the __builtin__ module every time the object is used.

# Global Optimizations - other __builtin__ objects

- We can still greatly increase the speed of stuff like the global class `int`.
- Pyrex performs a module lookup on the __builtin__ module every time the object is used.
  - Dynamic lookups are what makes Python slow...

# Global Optimizations - other __builtin__ objects

- We can still greatly increase the speed of stuff like the global class `int`.
- Pyrex performs a module lookup on the __builtin__ module every time the object is used.
  - Dynamic lookups are what makes Python slow...
- SageX caches ever __builtin__ lookup at module load time, and stores it in a global c variable.

# Global Optimizations - other __builtin__ objects

- We can still greatly increase the speed of stuff like the global class `int`.
- Pyrex performs a module lookup on the __builtin__ module every time the object is used.
  - Dynamic lookups are what makes Python slow...
- SageX caches ever __builtin__ lookup at module load time, and stores it in a global c variable.
  - In the same code, it makes sure it is a valid object in __builtin__ or it throws...

    ```
    undeclared name not builtin:  blah
    ```

## Future Work

# Future Work

- Merge Pyrex 0.9.5.1a changes

# Future Work

- Merge Pyrex 0.9.5.1a changes
- Generators

# Future Work

- Merge Pyrex 0.9.5.1a changes
- Generators
- Functional closures

# Future Work

- Merge Pyrex 0.9.5.1a changes
- Generators
- Functional closures
- cdef function type narrowing

# Future Work

- Merge Pyrex 0.9.5.1a changes
- Generators
- Functional closures
- cdef function type narrowing
- explicit coercions

# Future Work

- Merge Pyrex 0.9.5.1a changes
- Generators
- Functional closures
- cdef function type narrowing
- explicit coercions
- Non-refcounted extension types

# Future Work

- Merge Pyrex 0.9.5.1a changes
- Generators
- Functional closures
- cdef function type narrowing
- explicit coercions
- Non-refcounted extension types
- Further optimization

# Future Work

- Merge Pyrex 0.9.5.1a changes
- Generators
- Functional closures
- cdef function type narrowing
- explicit coercions
- Non-refcounted extension types
- Further optimization
- (Almost) anything else that doesn't work "out of the box."

# Future Work

- Merge Pyrex 0.9.5.1a changes
- Generators
- Functional closures
- cdef function type narrowing
- explicit coercions
- Non-refcounted extension types
- Further optimization
- (Almost) anything else that doesn't work "out of the box."
  - SageX should (almost) be a superset of the Python language.

# Overview

- Scanning.py tokenizes stream according to data in Lexicon.py

## Overview

- Scanning.py tokenizes stream according to data in Lexicon.py
- Parsing.py builds a Node tree from the tokens

# Overview

- Scanning.py tokenizes stream according to data in Lexicon.py
- Parsing.py builds a Node tree from the tokens
  - Python grammar

```
if_stmt    ::=   "if" expression ":" suite
.               ( "elif" expression ":" suite )*
.               ["else" ":" suite]
```

## Overview

- Scanning.py tokenizes stream according to data in Lexicon.py
- Parsing.py builds a Node tree from the tokens
  - Python grammar

```
if_stmt   ::=   "if" expression ":" suite
.              ( "elif" expression ":" suite )*
.              ["else" ":" suite]
```

  - Corresponding function

```
def p_if_statement(s):
    ...
```

## Overview

- Scanning.py tokenizes stream according to data in Lexicon.py
- Parsing.py builds a Node tree from the tokens
  - Python grammar

    ```
    if_stmt   ::=   "if" expression ":" suite
    .              ( "elif" expression ":" suite )*
    .              ["else" ":" suite]
    ```

  - Corresponding function

    ```
    def p_if_statement(s):
        ...
    ```

- Tree generates code.

# Other files

- Symtab.py keeps track of scopes, variable allocations, and declarations.

# Other files

- Symtab.py keeps track of scopes, variable allocations, and declarations.
- Code.py has lots of utilities for actually writing the c code.

# Other files

- Symtab.py keeps track of scopes, variable allocations, and declarations.
- Code.py has lots of utilities for actually writing the c code.
- TypeSlots.py, PyrexTypes.py, Naming.py contain definitions.

# C Code Generation

- Each node knows how to generate its c code

# C Code Generation

- Each node knows how to generate its c code
- Three main passes, called recursively

## C Code Generation

- Each node knows how to generate its c code
- Three main passes, called recursively
  - Analyse Declarations (what is being defined)

# C Code Generation

- Each node knows how to generate its c code
- Three main passes, called recursively
  - Analyse Declarations (what is being defined)
  - Analyse Expressions (determine types, fill in coercion nodes, determine needed temporary variables)

# C Code Generation

- Each node knows how to generate its c code
- Three main passes, called recursively
  - Analyse Declarations (what is being defined)
  - Analyse Expressions (determine types, fill in coercion nodes, determine needed temporary variables)
  - Generate Code (e.g. generate sub-expression code, do my stuff, dispose of sub-expression values)

# Code Generation in SAGE?

```
sage:  f = x^2 + 2*x + 3
```

# Code Generation in SAGE?

```
sage:  f = x^2 + 2*x + 3
sage:  print f.code(c.double)

double f(double x) {
    return (x+2)*x+3;
}
```

# Code Generation in SAGE?

```
sage:  f = x^2 + 2*x + 3
sage:  print f.code(python)

def f(x):
    return (x+2)*x+3
```

# Code Generation in SAGE?

```
sage:  f = x^2 + 2*x + 3
sage:  print f.code(c.gmp.mpz_t)

void f(mpz_t rop, mpz_t x) {
    mpz_add_si(rop, x, 2);
    mpz_mul(rop, rop, x);
    mpz_add_si(rop, x, 3);
}
```

## Code Generation in SAGE?

```
sage:  f = x^2 + 2*x + 3
sage:  print f.code(lisp)

(lambda (x)
    (+ (* (+ x 2) x) 3))
```

# Code Generation in SAGE?

```
sage:  f = x^2 + 2*x + 3
sage:  print f.code(sage.Integer)

cdef Integer f(Integer x) {
    cdef Integer r = PY_NEW(Integer)
    mpz_add_si(r.value, x.value, 2)
    mpz_mul(r.value, r.value, x.value)
    mpz_add_si(r.value, x.value, 3)
    return r
```