Python and Cython

Python

1. List Comprehensions

First, a list is defined by square brackets and can contain anything:

```
[1, PolynomialRing(QQ,2,'x,y'), 'a string']
```

```
[1, Multivariate Polynomial Ring in x, y over Rational Field, '
string']
```

```
primes20 = primes_first_n(20)
primes20
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59
67, 71]
```

List comprehensions make new lists from old. For example, the squares of the entries:

```
[i^2 for i in primes20]
```

```
[4, 9, 25, 49, 121, 169, 289, 361, 529, 841, 961, 1369, 1681, 1
2209, 2809, 3481, 3721, 4489, 5041]
```

List comprehensions are usually more readable than the equivalent functional programming:

```
map(lambda i:i^2, primes20)
```

```
[4, 9, 25, 49, 121, 169, 289, 361, 529, 841, 961, 1369, 1681, 1
2209, 2809, 3481, 3721, 4489, 5041]
```

The "if" clause lets you use only some list elements:

```
[p for i,p in enumerate(primes_first_20) if i%2 == 0]
```

```
[2, 5, 11, 17, 23, 31, 41, 47, 59, 67]
```

```
primes_first_20[0::2]    # can also be done with the slicing
operator
```

```
[2, 5, 11, 17, 23, 31, 41, 47, 59, 67]
```

2. Flow Control

I'll just mention the for loop. Note that blocks are marked by indentation!

```
for i in range(0,10):
    temporary_variable = 1
    print 'i = '+str(i)
print 'End of loop'
```

```
i = 0
i = 1
i = 2
i = 3
i = 4
i = 5
i = 6
i = 7
i = 8
i = 9
End of loop
```

3. Functions

```
def hello_world():
    print 'Hello, World!'
```

```
hello_world()
```

```
Hello, World!
```

4. Classes

```
class Foo(object):
    def bar(self):
        print('method bar() of class Foo.')
    j = 1    # data member
```

```
x = Foo()
x
```

```
<__main__.Foo object at 0x5745490>
```

```
x.bar()
```

```
method bar() of class Foo.
```

```
x.j
```

```
1
```

- Use tab-completion to discover attributes of objects

- Questionmark at the end -> online help
- Two questionmarks -> show source

```
p = 5
```

```
p.is_prime?
```

File: /home/vbraun/Code/sage.git/src/sage/rings/integer.pyx

Type: <type 'builtin_function_or_method'>

Definition: p.is_prime(proof=None)

Docstring:

Returns `True` if `self` is prime.

INPUT:

- `proof` – If False, use a strong pseudo-primality test. If True, use a provable primality test. If unset, use the default arithmetic proof flag.

Note

Integer primes are by definition *positive*! This is different than Magma, but the same as in PARI. See also the `is_irreducible()` method.

EXAMPLES:

```
sage: z = 2^31 - 1
sage: z.is_prime()
True
sage: z = 2^31
sage: z.is_prime()
False
sage: z = 7
sage: z.is_prime()
True
sage: z = -7
sage: z.is_prime()
False
sage: z.is_irreducible()
True
```

```
sage: z = 10^80 + 129
sage: z.is_prime(proof=False)
True
sage: z.is_prime(proof=True)
True
```

IMPLEMENTATION: Calls the PARI `isprime` function.

5. Inheritance

```
class Derived(Foo):
    def __init__(self, i):
        self.i = i
```

```
y = Derived(123)
y
```

```
<__main__.Derived object at 0x5745390>
```

```
y.i, y.j
```

```
(123, 1)
```

```
y.bar()
```

```
method bar() of class Foo.
```

6. Mutability and Immutability

Variables are labels for objects

```
x = [1, 2, 3]
y = x
x[1] = 'changed'
print(y)
```

```
[1, 'changed', 3]
```

A tuple is like a list but immutable

```
x = (1, 2, 3)     # equivalent to x = tuple([1, 2, 3])
y = x
x[1] = 'changed'
```

```
Traceback (click to the left of this block for traceback)
...
TypeError: 'tuple' object does not support item assignment
```

If you are writing a class, it is up to you to decide if it has a mutable or immutable interface. Sage sometimes implements both:

```
v = vector(ZZ, [1, 2, 3])
v[2] = 5
v
```

```
(1, 2, 5)
```

```
v.is_immutable()
```

```
False
```

```
v.set_immutable()
v[2] = 2
```

```
Traceback (click to the left of this block for traceback)
...
ValueError: vector is immutable; please change a copy instead (
copy())
```

Sage Extensions

There are a few Magma-inspired language extensions to Python

```
R.<x, y> = QQ[]
R
```

```
Multivariate Polynomial Ring in x, y over Rational Field
```

is equivalent to the Python commands

```
R = PolynomialRing(QQ, 2, names='x, y')
R.inject_variables()
R
```

```
Defining x, y
Multivariate Polynomial Ring in x, y over Rational Field
```

```
%python
1/2    # division in Python is C division
```

```
0
```

```
1/2
```

```
1/2
```

Cython

1. Speeding up Python

Here is a simple Python function:

```
def python_sum_0_99():
    s = 0
    for i in range(0,100):
        s += i
    return s
```

```
python_sum_0_99()
```

    4950

The `%cython` magic tells the worksheet that the cell contains Cython code:

```
%cython
def cython_sum_0_99():
    cdef int i, s
    s = 0
    for i in range(0,100):
        s += i
    return s
```

__home_vbr...6_code_sage45_spyx.c __home_vbr...ode_sage45_spy

Finally, we compare the two versions:

```
timeit('python_sum_0_99()')
```

    625 loops, best of 3: 108 µs per loop

```
timeit('cython_sum_0_99()')
```

    625 loops, best of 3: 89.6 ns per loop

2. Interface with C libraries

```
%cython
cdef extern from "math.h":
    double modf(double value, double* iptr)

def py_modf(x):
    cdef double iptr
    cdef double result = modf(x, &iptr)
    return (result, iptr)
```

__home_vbr...6_code_sage75_spyx.c __home_vbr...ode_sage75_spy

```
py_modf(pi)
```

    (0.14159265358979312, 3.0)

3. The C++ standard library

```
%cython
#clang c++

from libcpp.vector cimport vector
```

```
def using_vector(data):
    cdef vector[int] v
    for x in data:
        v.push_back(x)
    return v.size()
```

__home_vbr...ode_sage177_spyx.cpp __home_vbr...de_sage177_spy

```
using_vector([1, 3, 5])
```

3

4. Wrapping C++ Code