Leture 3a: Programming in Sage and Python

James Carlson

CIMAT Lectures

February 26, 2008

Outline

1. Numbers
   1.1 Arithmetic
   1.2 Types
   1.3 Complex field, ZZ, QQ, RR, CC
   1.4 Number fields
   1.5 Finite fields
   1.6 p-adic fields
2. Lists [2, 3, 5, 7]
3. Functions
   3.1 Lambda functions ; def f(x,y): ...
   3.2 for and while loops
   3.3 if C: then A, else B
4. Mathematical examples
   4.1 Factoring integers
   4.2 Quadratic residues
   4.3 Roots in finite fields

Arithmetic

```
sage: a + b
11
```

```
sage: a*b, a^b
(28, 2401)
sage: a/b, a % b
(7/4, 3)
sage: a//b, a % b
(1, 3)
```

```
sage: float(a)/b
1.75
sage: int(_)
1
```

Types

```
sage: parent(1)
Integer Ring
sage: parent(7/2)
Rational Field
sage: parent(3.1416)
Real Field with 53 bits of precision

sage: sqrt(-1)
I
sage: parent(_)
Symbolic Ring
sage: a, b = 1 + I, 1 - I
sage: a*b
(1 - I)*(I + 1)
sage: expand(_)
2
```

Complex field

```
sage: CC(sqrt(-1))
1.00000000000000*I
sage: parent(_)
Complex Field with 53 bits of precision

sage: a = 1 + sqrt(-5.0)
sage: a
1.00000000000000 + 2.23606797749979*I
sage: parent(a)
Complex Field with 53 bits of precision
```

Some common rings and fields: ZZ, QQ, RR, CC

Number Fields

```
sage: R.<x> = PolynomialRing(QQ)
sage: R
Univariate Polynomial Ring in x over Rational Field

sage: K.<i> = NumberField(x^2 + 1)
sage: K
Number Field in i with defining polynomial x^2 + 1

sage: i in K
True
sage: i*i
-1

sage: i.tab
... i.conjugate, i.norm, i.trace, ...
```

Number Fields ...

```
sage: a = 1 + i
sage: a.conjugate()
-i + 1
sage: a.norm()
2
sage: a.trace()
2
```

```
sage: K.galois_group()
Galois group PARI group [2, -1, 1, "S2"] of degree 2
of the number field Number Field in i with defining
polynomial x^2 + 1
```

Finite fields

```
sage: K = GF(11)
sage: K(5) + K(7), K(5)*K(7)
1, 2

sage: K.tab
... K.multiplicative_generator ...
sage: K.multiplicative_generator()
2

sage: L.<a> = GF(8)
sage: L
Finite Field in a of size 2^3
sage: for x in L:
    print x,",",
....:
0 , a , a^2 , a + 1 , a^2 + a , a^2 + a + 1 , a^2 + 1 , 1
```

p-adic numbers

```
sage: K = Qp(5)
sage: K
5-adic Field with capped relative precision 20

sage: 1/K(2)
3 + 2*5 + 2*5^2 + 2*5^3 + 2*5^4 + 2*5^5 + 2*5^6 + 2*5^7 + 2*5
2*5^9 + 2*5^10 + 2*5^11 + 2*5^12 + 2*5^13 + 2*5^14 + 2*5^15 +
2*5^16 + 2*5^17 + 2*5^18 + 2*5^19 + O(5^20)

sage: K(-1)
4 + 4*5 + 4*5^2 + 4*5^3 + 4*5^4 + 4*5^5 + 4*5^6 + 4*5^7 + 4*5
4*5^9 + 4*5^10 + 4*5^11 + 4*5^12 + 4*5^13 + 4*5^14 + 4*5^15 +
4*5^16 + 4*5^17 + 4*5^18 + 4*5^19 + O(5^20)

sage: _^2
1 + O(5^20)
```

Lists

```
sage: a = [1, 7, 2]; b = [4, 5]
sage: c = a + b; c
[1, 7, 2, 4, 5]
sage: c.sort(); c
[1, 2, 4, 5, 7]
```

```
sage: c.<tab>
```

```
c.append    c.extend    c.insert    c.remove    c.sort
c.count     c.index     c.pop       c.reverse
```

```
sage: c.append("foo"); c
[1, 2, 4, 5, 7, 'foo']
```

Manipulating lists

```
sage: c; c[0]
['foo', 7, 5, 4, 2, 1]; 'foo'

sage: c[0] = 11
sage: c
[11, 7, 5, 4, 2, 1]

sage: c[0:2]
[11, 7]

sage: c[2:]
[5, 4, 2, 1]

sage: c[:2]
[11, 7]
```

Constructing lists

```
sage: [ n^2 for n in range(2,10) ]
[4, 9, 16, 25, 36, 49, 64, 81]
```

```
sage: QR = [ n^2 % 11 for n in range(1,11) ]
sage: QR
[1, 4, 9, 5, 3, 3, 5, 9, 4, 1]
sage: 8 in QR
False
```

Lambda functions

```
sage: f = lambda x: x^2 % 11
sage: f(5)
3
```

```
sage: map(f, range(1,11))
[1, 4, 9, 5, 3, 3, 5, 9, 4, 1]
```

```
sage: from random import random
sage: rand = lambda n: int(n*random())
```

```
sage: for k in range(0,20):
....:     print rand(4),
....:
2 0 3 2 0 3 0 2 1 2 1 2 2 1 3 1 2 0 2 0
```

Functions

```
sage: def f(x):
....:     if x % 2 == 0:
....:        return x/2
....:     else:
....:        return rand(10)*x + rand(3)

sage: def run(f,a,N):
....:     while a > 1 and a < N:
....:         print a,
....:         a = f(a)

sage: run(f,10,1000)
10 5 25 27 54 27 190 95 191
sage: run(f,10,1000)
10 5 31 125 876 438 219
```

Loops

In the preceding examples we used loops to execute repetitive action.

There are two kinds of loops in Python / Sage.

The for loop:

```
for x in GF(7):
  print x, x^2
```

The while loop:

```
N = 12928
while N % 2 == 0:
  N = N/2
  print N,
```

Conditionals

We also used conditionals to make decisions:

```
if n % d == 0:
  n = n / d
  print d,
else:
  d = d + 1
```

Factoring

File "factor.sage"

```
def factor(n):
  d = 2
    while n > 1:
      if n % d == 0:
        n = n/d
        print d,
      else:
        d = d + 1
```

```
sage: load "factor.sage"
sage: factor(123456789)
3 3 3607 3803
```

Factoring 2

```
def factor2(n):
  d = 2
  F = [ ]
  while n > 1:
    if n % d == 0:
      n = n/d
      F.append(d)
    else:
      d = d + 1
  return F
```

```
sage: factor2(123456789)
[3, 3, 3607, 3803]
```

Quadratic residues

```
def unique(L):
  # return L without repeats
  M = [ ]
  for x in L:
    if x not in M:
      M.append(x)
  return M

def QR(N):
  # return list of quadratic residues mod N
  SQ = map( lambda x: x^2 % N, range(1,N) )
  SQ = unique(SQ)
  SQ.sort()
  return SQ

sage: QR(31)
```

Roots in finite fields

We want a function roots(f, q) which returns a list of roots of f(x) = 0 in GF(q).

The polynomial f(x) may have either integer or rational coefficients, so we have to clear denominators first.

```
def qq2zz(f):
  # clear denominators of f
  c = f.coeffs()
  d = map( lambda g: g.denom(), c)
  return lcm(d)*f
```

Roots in finite fields ...

```
def roots(f, q):
  # return list of roots of f in finite field of q elements
  K.<T> = GF(q)
  r = [ ]
  g = qq2zz(f).change_ring(K)
  for a in K:
    if g(a) == 0:
      r.append(a)
  return r
```

Roots in finite fields ...

```
def search(f,a,b,k):
  # search for roots of f in GF(p^k) for p in [a,b]
  for p in prime_range(a,b):
    rr = roots(f,p^k)
    if rr != [ ]:
      print p, rr
```

Roots in finite fields ...

```
sage: S.<u> = PolynomialRing(QQ)
sage: f = 2/5*u^5 + 3/7*u^2 + 1

sage: search(f, 2, 20, 2)
2 [1]
3 [2]
5 [0]
7 [0]
11 [6]
19 [4, 18*T + 11, 3, T + 10, 10]
```