A brief Python tutorial

Mladen Bestvina

May 9, 2007

1 Python as a calculator

To start python, type "python" on the command line. You should get something like

```
Python 2.5.1c1 (release25-maint, Apr 12 2007, 21:00:25)
[GCC 4.1.2 (Ubuntu 4.1.2-0ubuntu4)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Now you can start using python as a calculator. Here is a sample session.

```
>>> 2+3
5
>>> 7*6
42
>>> 8/3
2
>>> 8.0/3.0
2.6666666666666665
>>> 7%3
1
>>> divmod(7,3)
(2, 1)
>>> range(3,7)
[3, 4, 5, 6]
>>> 2**10
1024
```

```
>>> 2**100
1267650600228229401496703205376L
```

To exit the python mode type Ctrl-D. Note the difference between 8/3 and 8.0/3.0. Also observe that 7%3 means 7 mod 3 and that *divmod* returns both the quotient and the remainder. The function *range* returns all integers in the given interval (note that the first number is included but the last one is not). Exponentiation is denoted $**$ and arbitrarily long integers are allowed (when they are longer than usual, this is indicated by the letter L).

To use built-in mathematical functions you have to import them.

```
>>> from math import sqrt,log
>>> sqrt(2)
1.4142135623730951
>>> log(2)
0.69314718055994529
```

2 Python programs

It is often more convenient to put python commands in a file which is then imported (or executed). The following program lists all integers from 1 to 10.

```
for n in range(1,11):
    print n
```

When you type it in a file, do pay attention to indentations. They are an important part of the syntax. Say the file is called *listing.py*. To execute it, run "python listing.py" from the command line in the same directory. Here you also see an example of a "for loop". Note the colon and the indentation.

The following program lists all even numbers in the interval $1, 2, \cdots, 10$.

```
for n in range(1,11):
    if n%2==0:
        print n
```

In addition to the for loop, also note the "if" clause, along with the colon and the indentation. Observe the use of "==".

Save the following in a file called "GCD.py"

```
def gcd(a,b):
    if a<b:
        a,b=b,a
    if b==0:
        return a
    r=a%b
    return gcd(b,r)
```

Here we are defining a function *gcd* of two variables. Note how one swaps the values of a and b. The keyword "return" in effect defines the value of the function and exits the definition. To use this import the function.

```
>>> from GCD import gcd
>>> gcd(18,30)
6
```

3 Basic syntax

3.1 if clause

If you save the following code into a file and run it, what is the output?

```
x=6
y=8
if x>y:
    print ‘‘x is greater than y’’
else:
    print ‘‘x is not greater than y’’
```

3.2 for loop

Same question for

```
x=4
for i in range(1,x):
    print i
x=8
for i in range(1,x):
    print i
```

3.3 while

Same for

```
x=3
while x<7:
    x=x+1
    print x
```

3.4 Lists

Lists are most common data structures in python. They correspond to arrays in C and other languages.

```
>>> A=[2, 4, 6]
>>> B=range(1,7)
>>> B
[1, 2, 3, 4, 5, 6]
>>> A+B
[2, 4, 6, 1, 2, 3, 4, 5, 6]
>>> for i in range(3):
...     print A[i]
...
2
4
6
>>> A.append(8)
>>> A
[2, 4, 6, 8]
>>> A[-1]
8
>>> len(A)
4
>>> A+B[2:4]
[2, 4, 6, 8, 3, 4]
>>> (A+B)[2:4]
[6, 8]
```

You can see that a list can be defined by listing elements enclosed in brackets. The function *range* returns a list of integers in a specified interval. Lists can be concatenated using $A + B$, elements can be added, as in *A.append*, and individual elements can be retrieved (e.g. $A[0]$). Note that the first element is $A[0]$, second is $A[1]$ etc. The last one can be called with $A[-1]$. A sublist can be extracted with $A[i : j]$. The length of a list can be found using *len*.

3.5 List comprehension

What does the following code produce?

```
A=[1,2,4,8,9,10]
print [2*x for x in A]
print [2*x for x in A if x%2==0]
```

List comprehension is very useful. For example, the following one liner lists all primes between 2 and 1000.

```
[p for p in range(2,1000) if 0 not in [p%d for d in range(2,p)]]
```

Exercise. Write a function $primes(n)$ that lists all primes between 2 and n. Try to make the code run faster than in the above example (but it will no longer be a one liner!). For example, to test if p is prime you don't need to divide by all numbers between 2 and $p - 1$; it suffices to go to $\sqrt{p}$ (of course, you will have to import the square root function). Another speed improvement is to test divisibility by 2 and 3, and then by numbers congruent to 1 or 5 *mod* 6.

4 The sieve of Eratosthenes

There is another, much more efficient algorithm to list all primes from 2 to n. List all numbers

$$2, 3, \cdots, n$$

The first number, 2, is prime. Underline it. Then cross all multiples of 2, namely $4, 6, 8, \cdots$. The first uncrossed and ununderlined number is 3 and it is prime. Underline it and cross out all multiples of 3, namely $6, 9, 12, \cdots$

(some of these numbers are already crossed out). The first uncrossed and ununderlined number is 5. Underline it and cross out all multiples of 5, namely $10, 15, \cdots$. Continue in this fashion. The underlined numbers form the desired list of primes between 2 and n. Sketched below is how this works when $n = 9$, where we also put an arrow above the number whose multiples are being crossed out.

$$\overset{\downarrow}{\underline{2}} \quad 3 \quad \cancel{4} \quad 5 \quad \cancel{6} \quad 7 \quad \cancel{8} \quad 9$$

$$\underline{2} \quad \overset{\downarrow}{\underline{3}} \quad \cancel{4} \quad 5 \quad \cancel{6} \quad 7 \quad \cancel{8} \quad \cancel{9}$$

$$\underline{2} \quad \underline{3} \quad \cancel{4} \quad \overset{\downarrow}{\underline{5}} \quad \cancel{6} \quad 7 \quad \cancel{8} \quad \cancel{9}$$

$$\underline{2} \quad \underline{3} \quad \cancel{4} \quad \underline{5} \quad \cancel{6} \quad \overset{\downarrow}{\underline{7}} \quad \cancel{8} \quad \cancel{9}$$

Here is python code that lists primes < 10000.

```
def eratosthenes(n):
    d=range(n+1)
    for i in range(2,n+1):
        d[i]=1
    arrow=2
    while arrow<n:
        for k in range(2,n/arrow+1):
            d[k*arrow]=0
        arrow=arrow+1
        while arrow<n and d[arrow]==0:
            arrow=arrow+1
    for i in range(2,n):
        if d[i]==1:
            print i,

eratosthenes(10000)
```

On my laptop *eratosthenes* takes 0.1 seconds to list primes < 10000. For the same task, *primes* takes 15.3 seconds, while an improved version that goes to $\sqrt{p}$ takes 0.31 seconds. There is also an improvement of *eratosthenes*,

where one uses a data structure called *dictionary* instead of a list, that saves about 30% of time. Just replace the line

```
d=range(n+1)
```

by the line

```
d={}
```