Really quick Python tutorial

Victor Eijkhout

August 2004

1 The very basics

1.1 Running

- Interactively: `python`
- Call python: `python myfile.py`
- Make executable: `chmod +x myfile.py`
  file:
  ```
  #!/usr/bin/python
  ...
  ```
  also `#!/usr/bin/env python`

Use emacs: python mode

1.2 Code layout

Every statement on a line by itself: no semicolon needed.

With semicolon: more than one statement on a line.

Continuation by escaping line end.

Comments start with `#`.

2 Data

2.1 Numbers

Nothing too surprising. 2/3 gives an integer result 0

2.2 Strings

Strings in single or double quotes
otherwise largely like lists.

Concatenate: `'a'+"b"`

multiply `5*'word'`

2.3 Lists

List: `a=[1,2,'a','bcd']`

List run from 0: `a[1:3]` is `[2,'a']`

Slices: `a[2:]`, `a[:3]`, `a[:]`
(that last one is a copy, as opposed to `a`: copy of pointer)

Assign to slices: `a[2]=[3,4]`, `a[3:4]=[]`

Concatenate: `a+b[:3]`

Length: `len(a)`

Multidimensional: list of lists `a[3]=[4,5]`

2.3.1 Other list manipulation

`a.insert(i,x)` insert x before element i

`a.remove(x)` remove x; has to be present

`a.count(x)`, `a.index(x)` how many times present, where?

`a.append(x)`, `a.extend(L)` add at end of list

2.3.2 Tips and tricks

- A simple assignment `list1 = list2` does not create a second list: it only copies a pointer. To create `list1` as copy of `list2`, do `list1 = list2[:]`.
- Create empty list: `a= n*[0]`

3 Statements

White space is significant: in conditionals and loops and such, the clauses are indented.

```
if a>0:
    do_something
    and more
else:
    yet more
```

In emacs, this indentation is done automatically

return next line with proper indentation
delete to the previous indentation level
tab the the next level, but only if that is possible:

```
if a>0:
    do_something
    and more
after
```

the `after` can be tabbed to belong in the conditional.

3.1 Control structures

Conditional:

```
if a>0:
    print "yes"
elif b<0:
    print "no"
else:
    print "hm"
```

Also

```
if a>0: print a
```

Loop:

```
for w in words:
    print w
```

range over array elements. Numerical index:

```
for i in range(4):
    f(i)
```

ranges over indix array `[0,1,2,3]`. Also `range(1,4)` is `[1,2,3]`; `range(4,1,-1)` is `[4,3,2]`.

While loop:

```
while p>0:
    p = prev[p]
```

`break` and `continue` statements

3.2 Functions

Define

```
def fact(n):
    if n==0: return 1
    else:
        return n*fact(n-1)
```

All variables local unless

```
    global x
```

included

3.3 Input/output

Use `raw_input`; the `input` command evaluates its input (which can be dangerous, since you can sneak system commands in there).

Output with `print`. This automatically inserts a newline; prevent that with `print,`. For precise control over spacing and newlines:

```
import sys
sys.stdout.write(<something>)
```

This only works with strings
convert `str(23)`

```
try:
    a = raw_input()
    print "line:",a
except (EOFError):
    break
```

Contents

1 **Some examples** *1*
1.1 *Decision timing 1*
1.2 *A manufacturing problem 2*
1.3 *The stagecoach problem 3*
1.4 *Traveling salesman 7*
2 **Discussion** *7*