Python

Victor Eijkhout

Notes for CS 594 – Fall 2004

The very basics

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

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 #.

Data

Numbers

Booooring

2/3 is zero; `1.*2/3` float

Strings

Strings in single or double quotes
otherwise largely like lists.
Concatenate: `'a'+"b"`
multiply `5*'word'`

Lists

The basics

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]`

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

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]`

Statements

White space

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

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

Python support in emacs

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.

Control structures

Conditionals

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

Also

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

Loops

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

range over array elements. Numerical index:

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

ranges over index 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

Functions

Define

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

All variables local unless

```
    global x
```

included

I/O

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: `str(23)` et cetera for conversion

EOF

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