Lecture 31

- Scripting Languages
  - Shells (bash, ksh93, awk etc)
  - Programing Languages (C++, Java, SML)
- Easy to use, interpreted, dynamic, easy to glue existing code
- Tcl, Perl, Python (the troica)
- Javascript, PHP, Ruby

Perl (L.Wall)

- There is more than one way to do it
- Natural language, designed to evolve
- Duct tape of the internet
- Laziness, impatience and hubris
- So by and large, those computer scientists who can hold their nose long enough to get the cheese into their mouths find the taste bearable.

Tcl (J.Ousterhout)

- Origins: IC design (UofBerkeley) (1989-1990)
- Embeddable language (Tool Commad Lang)
- Glue existing components
- Tk (user interface toolkit)
  - perl_tk, python_tk

```
proc power {base p} {
   set result 1
   while {$p > 0} {
      set result [expr $result *
      set p [expr $p - 1]
   }
   return $result
}
```


Python (www.python.org)

- 1990 Guido Van Rossum
- Influenced by ABC (teaching language), Modula 3
- Good support for interfacing with C/C++
  - easy to extend using modules
- A really nice language (IMHO much nicer than Perl, Tcl)
- Named after Monty Python
- A xmas project :-)

Pros & Cons

- Pros
  - Fast rapid prototyping (no compile-link-run) – interactive
  - Easy to read
- Cons
  - No compile time error analysis and type declaration
  - Slow (10-100 times slower than Lisp, SML)
- Scheme with better libraries, cleaned up Perl

How does it look

```
def invert(table):
    index = {}              # empty dictionary
    for key in table.keys():
        value = table[key]
        if not index.has_key(value):
            index[value] = [] # empty list
        index[value].append(key)
    return index
```

Identation used
for block structure
(a controversial feature)

Using the function

```
>>> phonebook = {'guido': 4127, 'sjoerd': 4127, 'jack': 4098}
>>> phonebook['dcab'] = 4147          # add an entry
>>> inverted_phonebook = invert(phonebook)
>>> print inverted_phonebook
{4098: ['jack'], 4127: ['guido', 'sjoerd'], 4147: ['dcab']}
```


Lists

```
>>> a = [66.6, 333, 333, 1, 1234.5]
>>> print a.count(333), a.count(66.6), a.count('x')
2 1 0
>>> a.insert(2, -1)
>>> a.append(333)
>>> a
[66.6, 333, -1, 333, 1, 1234.5, 333]
>>> a.index(333)
1
>>> a.remove(333)
>>> a
[66.6, -1, 333, 1, 1234.5, 333]
```

```
>>> a.reverse()
>>> a
[333, 1234.5, 1, 333, -1, 66.6]
>>> a.sort()
>>> a
[-1, 1, 66.6, 333, 333, 1234.5]
```


Functional tools I

```
>>> def f(x): return x % 2 != 0 and x % 3 != 0
>>> filter(f, range(2, 25))
[5, 7, 11, 13, 17, 19, 23]

>>> def cube(x): return x*x*x
...
>>> map(cube, range(1, 11))
[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]

>>> seq = range(8)
>>> def add(x, y): return x+y
>>> map(add, seq, seq)
[0, 2, 4, 6, 8, 10, 12, 14]
```

Functional Tools II

```
>>> freshfruit = ['  banana', '  loganberry ', 'passion fruit  ']
>>> [weapon.strip() for weapon in freshfruit]
['banana', 'loganberry', 'passion fruit']
>>> vec = [2, 4, 6]
>>> [3*x for x in vec]
[6, 12, 18]
>>> [3*x for x in vec if x > 3]
[12, 18]
>>> [3*x for x in vec if x < 2]
[]
```


Modules

```
# Fibonacci numbers module

def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while b < n:
        print b,
        a, b = b, a+b

def fib2(n): # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result
```

fib.py

functions are not directly accessible

import fib
then
>>> fib.fib(10)

Can assign to local name:

fib = fib.fib

Persistant objects

- pickle
- pickle.dump(x, f)
- x = pickle.load(f)
- Standard module to save/load complicated data types – easy to extend for user defined types