python - easy deploying

Christian Kniep

International Center of Applied Technologies Bandung

11. August 2010

Table of content

1. Introduction
   - Low vs High Level
   - Interpreter / Scripting Languages
2. Step By Step
   - Executables
   - Strings
3. Comparisons
   - if-then-else
   - Algebra
   - Digits
4. Math
   - Basics
5. Librarys
   - math

Once upon a time...

- 'ugly' assembler had to be used (e.g. MIPS)

```
.data # define some varbiables
# the string we want printed
out:  .asciiz "Hello World"

.text # the program
main: li $v0, 4   # cmd-reg to cmd 4 ('print')
      la $a0, out # set out as the 1st arg
      syscall     # execute the cmd
      li $v0, 10  # cmd-reg to cmd 10 ('exit')
      syscall     # execute the cmd
```

don’t get me wrong

- Assembler is the closest,directest way to program a CPU

don't get me wrong

- Assembler is the closest,directest way to program a CPU

⇒ so its the fastest code you can write

don’t get me wrong

- Assembler is the closest,directest way to program a CPU

⇒ so its the fastest code you can write

- you might just want to switch on the coffee-maker

don't get me wrong

- Assembler is the closest,directest way to program a CPU

⇒ so its the fastest code you can write

- you might just want to switch on the coffee-maker

⇒ you don't care if it takes 2 nano- or 200 milliseconds

High Level Programming

- it doesn’t mean you have to have a high level to program it

High Level Programming

- it doesn't mean you have to have a high level to program it
- it the opposite: you only have to handle abstracted commands the assembler code will be created for you

High Level Programming

- it doesn't mean you have to have a high level to program it
- it the opposite: you only have to handle abstracted commands the assembler code will be created for you
- checkout 'hello world' in C

```
#include <stdio.h>

main()
{
 printf("Hello␣World␣\n");
}
```

python-Programming

- python is even more high level then c

```
print "hello␣world"
```

python-Programming

- python is even more high level then c
  - doesn't care what type you are using

```
print "hello␣world"
```

python-Programming

- python is even more high level then c
  - doesn’t care what type you are using
  - the syntax is intuitive

```
print ”hello␣world”
```

python-Programming

- python is even more high level then c
  - doesn’t care what type you are using
  - the syntax is intuitive
  - its an interpreter language, so you don’t have to compile it
  - it’s build from scratch, so there is no(t much) historical payload

```
print "hello␣world"
```

whats this suppose to mean?

- you don’t have to create an binary file

whats this suppose to mean?

- you don't have to create an binary file
- the programm is evaluated linewise

whats this suppose to mean?

- you don’t have to create an binary file
- the programm is evaluated linewise

⇒ so you are able to get a prompt

```
$ python
>>> print ”Hello World”
Hello World
```

- So if you see >>> I am in live-mode

whats this suppose to mean?

- you don’t have to create an binary file
- the programm is evaluated linewise

⇒ so you are able to get a prompt

```
$ python
>>> print "Hello World"
 Hello World
```

- So if you see >>> I am in live-mode
- The indentation is important!

```
>>>     print 'hello'
  File "<stdin>", line 1
    print 'hello'
    ^

 IndentationError: unexpected indent
```

Hello World #2

Now we want to create a little script that can be executed.

- We need a header

```
#! python

 print ”Hello World”
```

Hello World #2

Now we want to create a little script that can be executed.

- We need a header

  ```
  #! python

  print "Hello World"
  ```

- The header simply says what language do we use.
  So the OS could decide what program to use

Execute it

- If you just have a textfile (without execute-permission)

```
$ python helloWorld.py
 Hello World
```

Execute it

- If you just have a textfile (without execute-permission)

```
$ python helloWorld.py
Hello World
```

- Due to the header the OS will know that it should use python:

```
$ chmod +x helloWorld.py
$ ./helloWorld.py
Hello World
```

Work with strings

- concatenate

```
>>> print "1" + "1"
11
```

Work with strings

- concatenate

```
>>> print "1" + "1"
11
```

- Strings with linebreaks

```
>>> x= """
1st line
2nd line
"""
>>> print x
1st line
2nd line
```

Work with strings #2

- multiply

```
>>> print ”X”∗10
XXXXXXXXXX
```

Work with strings #2

- multiply

```
>>> print ”X”∗10
XXXXXXXXXX
```

- formated output

```
>>> print ”decimal: %d” % 1
decimal: 1
>>> print ”dec: %d – %d – %d” % (1,2,3)
1 – 2 – 3
>>> print ”str: %s” % ”some string”
str: some string
```

Work with strings #3

- more formated output

```
>>> print "%-10s # %-10s" % ("Christian","Kniep")
Christian  # Kniep
>>> print "%-10s # %-10s" % ("Han","Solo")
Han        # Solo
```

Work with strings #3

- more formated output

```
>>> print "%-10s # %-10s" % ("Christian","Kniep")
Christian  # Kniep
>>> print "%-10s # %-10s" % ("Han","Solo")
Han        # Solo
```

- good for useability, clearly arranged

some other commands

- adjust strings

```
>>> print hello.center(10)
   hello
>>> print "hello".center(10,"-")
--hello---
>>> print "hello".ljust(10,"-")
hello-----
>>> print "hello".rjust(10,"-")
-----hello
```

some other commands #2

- a quick glance

```
>>> len(X)
10
>>> x = "Hallo Welt"
>>> len(x)
10
>>> x.find("l")
2
>>> x.count("l")
3
...
```

use if

- simply type:

```
>>> if True:
...   print 'here its true'
...   print 'here also'
... else:
...   print 'now the false part'
...
here its true
here also
>>>
```

Whats the shifting and the dots about?

- In python the intendtion is part of the language.

Whats the shifting and the dots about?

- In python the intendtion is part of the language.
- e.g. in the if-statement you have to shift the command-block at least one char and stick to it

Whats the shifting and the dots about?

- In python the intendtion is part of the language.
- e.g. in the if-statement you have to shift the command-block at least one char and stick to it
- otherwise the interpreter will throw an error

```
>>> if True:
...     print 'right intention'
...   print 'now I will get an error'
  File "<stdin>", line 3
    print 'now I will get an error'
                                  ^
IndentationError: unindent does not match
any outer indentation level
```

Whats the shifting and the dots about?

- In python the intendtion is part of the language.
- e.g. in the if-statement you have to shift the command-block at least one char and stick to it
- otherwise the interpreter will throw an error

```
>>> if True:
...     print 'right intention'
...   print 'now I will get an error'
  File "<stdin>", line 3
    print 'now I will get an error'
                                  ^
IndentationError: unindent does not match
any outer indentation level
```

- please note that python always shows you the exact error and the exact line

Basic Algebra

But back to the comparision-topic. Its kind of awkward to use True and False in an if-statement. Actually you will use complex conditions.

- 3 operators included

Basic Algebra

But back to the comparision-topic. Its kind of awkward to use True and False in an if-statement. Actually you will use complex conditions.

- 3 operators included

AND True and True = True

- True and False = False

Basic Algebra

But back to the comparision-topic. Its kind of awkward to use True and False in an if-statement. Actually you will use complex conditions.

- 3 operators included
  - AND True and True $=$ True
    - True and False $=$ False
  - OR True or True $=$ True
    - True or False $=$ True
    - False or False $=$ False
  - NOT not True $=$ False
    - not False $=$ True

Basic Algebra

But back to the comparision-topic. Its kind of awkward to use True and False in an if-statement. Actually you will use complex conditions.

- 3 operators included
  - AND True and True = True
    - True and False = False
  - OR True or True = True
    - True or False = True
    - False or False = False
  - NOT not True = False
    - not False = True
- mix em!

  True and (False or not False) = ?

Digits

And these conditions will be some comparision...

- basic digits

```
>>> 1==1
True
>>> 1!=1
False
>>> 1>1
False
>>> 1>=1
True
>>> 1<1
False
>>> 1<=1
True
```

Strings

- compare strings

```
>>> "X"=="X"
True
>>> "X"<="X"
True
>>> "Y"<="X"
False
>>> "Hello World".startswith("H")
True
```

types

- compare types

```
>>> type("X")
<type 'str'>
>>> type(1)
<type 'int'>
>>> type(1.0)
<type 'float'>
>>> type("X") == type(1)
False
```

Hello World in calculation

- Simple example...

```
>>> x = 1
>>> y = 8
>>> print x + y
9
```

- advanced example...

```
>>> sqrt(9)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'sqrt' is not defined
```

endless opportunities

- normal import

```
>>> import math
>>> math.sqrt(9)
3.0
```

endless opportunities

- normal import

```
>>> import math
>>> math.sqrt(9)
3.0
```

- import exclusiv commands

```
>>> from math import sqrt
>>> sqrt(9)
3.0
```

endless opportunities

- normal import

```
>>> import math
>>> math.sqrt(9)
3.0
```

- import exclusiv commands

```
>>> from math import sqrt
>>> sqrt(9)
3.0
```

- import all commands

```
>>> from math import *
>>> ceil(3.6)
4.0
```

Questions?

- Any questions?

Questions?

- Any questions?
- What about OOP