UNIVERSITY
OF APPLIED SCIENCES
MÜNCHEN

Home Automation with Python on Raspberry Pi

Introduction into

python™

Prof. Dr. Jochen Hertle

The Python Community

- Open source community host: https://www.python.org/
- Documentation: https://docs.python.org/2.7/ (resp. /3.4/)
- Tutorial: https://docs.python.org/2.7/tutorial/index.html

- TIOBE Index
  - http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html
  - 03/2015: rank 8 (after C, Java, Objective-C, C++, C#, PHP, JavaScript)
- The Transparent Language Popularity Index
  - http://lang-index.sourceforge.net/
  - 07/2013: rank 2 within scripting languages after PHP
- PYPL PopularitY of Programming Language index
  - Focusses on Web technologies
  - http://pypl.github.io/PYPL.html
  - 03/2015: rank 3 (after Java, PHP)

Python Versions

- Actual version 3.4.3 (February 2015)
- 2.7 pre-installed on most Linux systems
  - Large existing 2.7 compatible code base
- 3.4.x is not backward compatible to 2.7 !!
- Recommendation: Write code that runs with 2.7 interpreter and is forward compatible to 3.4
  - Critical issues are `print` statement and integer `division`
    - 2.7: print as a statement: `print "Hello", name`
    - 3.4: print as a function:
      ```
      def print(*args, sep=' ', end='\n', file=None)
      print("Hello", name)
      ```
    - 2.7: `1/2 -> 0` (integer division)
    - 3.4: `1/2 -> 0.5` (float division, integer division is `1//2 -> 0`)
  - Solution for 2.7: `from __future__ import print_function, division`
  - For further issues see https://docs.python.org/2/library/__future__.html

Python IDEs

- Eclipse with pydev plugin from http://pydev.org/updates
  - Including debugging and unit-testing support
  - For Linux or Windows (just needs Java installed)
  - Recommended IDE for this course
- Spyder (https://pythonhosted.org/spyder/)
  - Comes with Python packages numpy and scipy for scientific computing
- Anaconda (http://continuum.io/downloads)
  - Comes with >190 Python packages
  - Includes Spyder
  - Anaconda is installed on all Windows PCs in FK07 labs
- WinPython (http://winpython.sourceforge.net/ only for Windows)
  - Portable distribution (i.e., everything is in one folder which can be copied or kept on an USB stick)
  - Different configurations can be held on one machine in different folders (i.e. 64bit, 32bit, Python 2.7 and 3.4, etc.)
  - Includes Spyder

Running Python Scripts

- Within the IDE
  - Only for scripts that do not need access to special target hardware (camera, GPIOs,…)
  - Just press button 'run'
  - Output of print statements and error messages appear in a special console window
- On the target
  - In the (ssh-)terminal type
    - `> python xy.py` (execution with user rights)
    - `> sudo python xy.py` (execution with root rights)
  - Default configuration on a Linux system:
    - `python` is `/usr/bin/python` which is a link to `/usr/bin/python2.7`
  - Alternative (used e.g. to start python processes from other scripts)
    - First line in script xy.py is `#!/usr/bin/python`
    - Script `xy.py` must be executable (e.g. `> chmod +x xy.py`)
    - Execute `xy.py` from the command line or from other scripts

Python Performance

- Python is intended for system integration and controlling programs on a higher level
- Python relies on libraries that are built in C-code
  - E.g., `numpy, scipy, cv2, …`
- Developer's rule:
  - Do not implement heavy computations in Python
    - long for/while-loops, array operations, …
  - Instead, find the appropriate library for the task
- Starting up Python is also time consuming
  1. Starting the python interpreter process
  2. Import modules
  3. Checking syntax of imported modules and the script
  4. Finally executing the script
- Use strategies to start the interpreter early (not at execution time)
  - Daemon processes
  - Python – apache integration (`mod-python`)

Indentation

- A block of statements is defined through the same indentation
  - 'same' = same number of whitespace characters to the left
  - Note: Do not use TAB, only ␣ !! (default for most Python IDEs)
- Start of a block is a colon :
- Examples:

```
if n == 5:
    print("We got a 5!")
    n = n+1
else:
    print("It's not a 5!!", file=sys.stderr)
    raise Not5Exception


def JoinNames(firstname, lastname):
    name = lastname + ", " + firstname
    print(name)
    ...
```

- Benefit: Less { } and better code readability

Types

- Python has got data types but it is not a strongly typed language
  - Example of a sequence of statements that execute without error:

```
N = 5
N += 1.2
N = "hello"
```

- Basic types
  - Integers: `int` (32 bit)
  - Long integers: `long` (Python version 3.x: no distinction between int and long)
  - Floating point numbers: `float`
  - Booleans: `bool`
  - Character strings: `str`
  - Unicode character strings: `unicode`
  - Lists: `list`
  - Tuples: `tuple`
  - Associative arrays (dictionaries): `dict`
  - Sets: `set`

Types - Examples

```
long: 100L
float: 1.2e-8
bool: True, False
str: 'hello', "hello", """hello""" (multiline string)
unicode: u"hello"
tuple: (1,4,3)
list: [4,7,5,'a']
dict: {'a': 3, 'b': "test", 'c': 3.141}
set: {5,7,10,3}
```

- Conversion examples:

```
list((1,4,3)) -> [1,4,3]
str(15) -> '15'
int('42') -> 42
```

String formatting

- `string.format(…)`
- See https://docs.python.org/2/library/string.html
- Examples:

```
"One {0} please!".format("beer") -> "One beer please!"
"{0}, {1}, {0}".format("first", "next") -> "first, next, first"
"Keyword {key}".format(key="Python") -> "Keyword Python"
"Float {0:f}".format(5) -> "Float 5.000000"
"Float {0:.2f} {1:.3e}".format(3,0.4) -> "Float 3.00 4.000e-01"
"List {0[2]}".format([2,3,4]) -> "List 4"
"X: {co[0]}, Y: {co[1]}".format(co=(3,5)) -> "X: 3, Y: 5"
"Object {0.weight}".format(M) -> "Object 82" (if M.weight = 82)
"hex: {0:x}".format(42) -> "hex: 0x2a"
"percent: {0:.2%}".format(1.0/8) -> "percent: 12.50%"
"{{x}}, {{{0}}}".format("y") -> "{x}, {y}"
```

Numpy Array

- Package numpy provides special functions for numeric computation
- Data type numpy.array has very powerful features for fast array manipulation
- Numpy arrays can be used to store images taken with the Raspberry Pi Camera and do computer vision computations
  - Example:

```
# assume img is the numpy array holding the RGB image
numpy.shape(img) # returns e.g. (1920,1080,3)
red = img[:,:,0] # red is the red channel of the image
numpy.shape(red) # returns (1920,1080)
```

- See `demo_numpy.py` for further examples

Debugging

- Debugging locally in the IDE
  - Only for scripts that do not need access to special target hardware (camera, GPIOs,…)
  - Set breakpoints: right-clicking the mouse at the beginning of a line
  - Start debugger
  - (see Eclipse demo)
- Debugging on the target
  1. Use the remote debugging feature of pydev within Eclipse
     - http://pydev.org/manual_adv_remote_debugger.html
     - https://sites.google.com/site/programmersnotebook/remote-development-of-python-scripts-on-raspberry-pi-with-eclipse
  2. Debugging on the target with the python built-in debugger

```
import pdb
....
pdb.set_trace()
```
(breakpoint)

see https://docs.python.org/2/library/pdb.html

commands: https://docs.python.org/2/library/pdb.html#debugger-commands

Python Demo Scripts

- List
- Dict
- Numpy array
- Function definitions
- Classes
- Exception handling
- Debugging

Unit-tests

- Unit-tests are classes derived from base class TestCase

```
import unittest
class TestCaseXY(unittest.TestCase):
    def setUp(self):
        ...
    def tearDown(self):
        ...
    def testCase1(self):
        self.assertEqual(…)
    def testCase2(self):
        self.assertRaises(…)
```

- Eclipse pydev: 'run as unit-tests'
  - Demo file `demo_unittest.py`
- On target: `> python -m unittest discover`

Mod_python

- Module for apache web server
- Python replacement for PHP
  - Needs installation of mod_python and configuration of apache config files
  - See http://webpython.codepoint.net/mod_python_tutorial
  - Very fast server-side python script execution
- Two modes of usage
  - Publisher mode
    - files are *.py scripts
    - Return value of functions is HTML text (as a string)
    - Function names are part of the URI
    - Default function is `index()`
  - Python Server Pages (PSP mode)
    - Files are *.psp
    - Content of files is HTML text with embedded Python code
    - Embedding with `<% … python code … %>`
- See mod_python demo

Python Webservices

- Use publisher mode of mod_python for server scripts
  - Returning e.g. json-encoded data
- Issue Javascript AJAX calls from client
  - E.g., $.ajax construct from jquery library
  - Use json encoding to send data to server
- See webservice demo files

Python socket server and client

- Used for inter-process-communication (IPC) or client-server communication over networks
- Base class is `asyncore.dispatcher`
- Global function `asyncore.loop()` in module `asyncore` is polling the handler functions that send an receive data
- See demo implementation:
  - `SocketEchoServer.py`
  - `SocketClient.py`
  - (run processes in different python consoles)

Websockets

- Websockets are part of HTML 5
  - Most actual browsers do support websockets
- Websockets allow for fast communication between client and server without the overhead of the HTTP protocol
- Websocket connections are initiated by the client
- Connections can be closed by the server or by the client
- Messages can be sent from client to server or from server to client
  - E.g., the server can inform the client (browser) about events
- With the mod_python extension for apache, the websocket handler on the server side can be written in Python
  - Special entries in apache2.config file needed
  - http://pywebsocket.googlecode.com/svn/trunk/src/mod_pywebsocket/__init__.py
- On the client side, websocket handlers are written in Javascript
- See files `echo_wsh.py`, `websocket.html` and `websocket.js`

Python Daemon Processes on Linux Systems

- Shell scripts in `/etc/init.d`
- Required header in shell script

```
### BEGIN INIT INFO
# Provides:          test_daemon
# Required-Start:    $remote_fs $syslog mysql
# Required-Stop:     $remote_fs $syslog mysql
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: starts and stops the test_daemon.py
# Description:       starts and stops the test_daemon.py
### END INIT INFO
```

- `Provides:` facility name used for dependencies
- run-levels `0,...,  6` (see http://de.wikipedia.org/wiki/Runlevel)
- Activation: `> sudo update-rc.d xxx.sh defaults`
- Usage: `> sudo service xxx.sh start/stop/restart`

Stopping Python Daemon Processes

- The Python interpreter will catch most OS signals and terminate
  - HUP (1), ABRT (6), KILL (9), TERM (15), STOP (17), etc.
  - No controlled bringing down of Python script possible (e.g.: finish a write to the data base and close connection to DB)
- Only one signal is passed through to the Python script: INT (2)
  - INT = keyboard interrupt, Ctrl-C
- Catching the INT signal is recommended for all Python daemon processes
- Stopping a daemon process using the shell command:
  `kill -s INT <PID>`
  - Strategy: During `init()` of Python process, write a shell command file that contains this command and the correct `PID`
- Establish a signal handler within the Python daemon script
  - Controlled bringing down after having received the INT signal