Python and FME

FME UC 2014 Training Session
Introduction

- Employed at con terra GmbH since 2009
- Spatial ETL Professional Services
- FME Certified Trainer since 2010
- FME Certified Professional since 2013
Agenda

- Introduction
- 4 Timeslots
  - 9am - 10.30am
  - 10.45am - 12.15pm
  - 1.15pm - 2.45pm
  - 3pm - 4.30pm
- Course Content
- Work environment
Resources

- http://fme.ly/fmeuecdns
- http://www.safe.com/uctraining
Course content forenoon

- Chapter 0: Introduction to Python
- Chapter 1: Variables, Lists, Dictionaries
- Chapter 2: Loops and conditions
- Chapter 3: FME Workspaces and Python
Course content afternoon

- Chapter 4: Configuring Eclipse
- Chapter 5: Python WorkspaceRunner
- Chapter 6: Custom Format Reader
- Chapter 7: Debugging
Hardware

- Your computer
- Printed exercise manual
- Personal EC2 image
Chapter 0

- Introduction to Python
Python

- Python is a scripting language.
  - But Object oriented
  - No compiling or linking
  - Fast ("quick&dirty") programming and prototyping
- Name: Developer van Rossum is a huge fan of Monty Python’s Flying Circus
  - IDE IDLE=> Eric Idle
  - Many references in the documentation
Why Python?

- Free, powerful and flexible
- Platform independent
- Automatic Garbage Collecting
- Capable of being integrated
  - e.g. FME, ArcGIS, Blender
- Extensive documentation
  - www.python.org
Python version havoc

- Latest versions - Python 2.7.7 und Python 3.4.1
- Python 3.x and 2.x are incompatible
  - Most libraries available for both versions
  - 2.x support and (security) bugfixes until 2020
- FME 2014 supports Python 2.7 ( - 2.5)
- More details on www.python.org
Interactive Shell

- **Start > Programs > Python 2.7 > Python (command line)**
- Execute single statements
- Not very comfortable
Python IDE – Integrated Development Environment

- IDLE – Shell with advanced functionality:
  - Included in the Python default distribution package
  - Interactive- and Script window
  - Code completion
  - Colorizing
  - Debugging
  - Call tips
More IDEs

- PythonWin
  - Editor available for Windows
- PyDev for Eclipse
  - Very helpful when coding gets more complex
- Simple Texteditor (e.g. Notepad)
  - Safe file with suffix .py
  - No debugging or colorizing
  - No shell
Interactive mode

- One line = one statement
- Exceptions:
  - Separate several commands in one line with semicolon
  - One command over several lines:
    - With Backslash ("\")
    - Strings with triple quotation marks
    - Using brackets

```python
>>> a = 1+2; print a
3
>>> a = 3 + \ 4
>>> print a
7
>>> print """"One String over two lines!"""
One String over two lines!
>>> a = (10 + 20)
>>> print a
30
```
Python preface

- Case-sensitive
- Variable and module names have to begin with a alphabetic character
  - Any character, numbers or underscores "_" can follow
  - No special characters (e.g. / \ § $ % &)
- You can use both single (') and double (") quotes
- Comments with # or """A comment"""" (triple quotes)
- dir(object) shows all properties of an object
Reserved keywords

- and
- assert
- break
- class
- continue
- def
- del
- elif
- else
- except
- exec
- finally
- for
- from
- global
- if
- import
- In
- is
- lambda
- not
- or
- pass
- print
- raise
- return
- try
- while
- yield
Exercise 1.1 and 1.2

- Get to know the Python commandline and IDLE
Chapter 1

- Variables in Python
Course content forenoon

- Chapter 0: Introduction to Python
- Chapter 1: Variables, Lists, Dictionaries
- Chapter 2: Loops and conditions
- Chapter 3: FME Workspaces and Python
Variables

- Container for everything you want to reuse
- Have a name
- Point to an address in your computer's memory
- Access the value via the name
- No declaration in Python
  - Dynamic typing

```python
a = 31  # Variable „a“ is a number (Integer)
name = „Don“  # Variable „name“ is a String
feature = fmeobjects.FMEFeature()  # Variable „Feature“ ist an object
```
Numbers

- Integer, Long, Double...
- Basic arithmetics +, -, *, /
- Modulo-Operator %
- Exponent **

```python
>>> 3+3
6
>>> 3-3
0
>>> 3*3
9
>>> a=3.0;b=3.0
>>> print a/b
1.0
#Exponent
>>> 3**3
27
#Modulo operator
>>> 13%5
3
```
More on numbers

- Many mathematical functions available via module math.
  - A default module, but has to be imported
- Examples
  - Square root
  - Constants, e.g. Pi
- Show properties of math:
  - dir(math)

```python
>>> import math
>>> math.sqrt(2)
1.4142135623730951
>>> math.pi
3.1415926535897931
```
Strings

- Enclose in quotation marks
  - text = “Hello World“
- Escape characters with backslash
- Control characters
  - "\n" -> New line
  - "\t" -> Tabulator
- Mark raw strings with r:
  - path_to_file = r"E:\Europe\cities.dxf"

>>> print "Hello World"
Hello World
>>> print 'Hello World'
Hello World
>>> print "Hello \nWorld"
Hello "World"
>>> print "Hello \nWorld"
Hello
World
>>> print "Hello \tWorld"
Hello     World
>>> print r"Hello \tWorld"
Hello \tWorld
More on Strings

- Use three quotation marks to go over multiple lines
- Just concatenate strings with “+”
- You can even multiply strings with “*”
- Use the built-in function `len()` to get the length of a string
- Strings are indexed, use square brackets to access parts
  - `My_string[start_pos:end_pos]`
Strings – Examples

```python
>>> print '''More than one line'''
More than one line

>>> print 'Part one,' + ' Part two'
Part one, Part two

>>> print "FME!" * 3
FME! FME! FME!

>>> name = "Tino"
>>> print len(name)
4

>>> greeting = "Hello World!"
>>> print greeting[4]
o
>>> print greeting[3:7]
lo W
>>> print greeting[3:-2]
lo Wor
>>> print greeting[:7]
Hello W
>>> print greeting[3.0:5]
Traceback (most recent call last):
  File "<pyshell#47>", line 1, in ?
    print greeting[3.0:5]
TypeError: slice indices must be integers
```
Lists can contain elements of different types
- `mylist = ['Tino', 1, 3.5]`

Recursive lists – lists in lists

Creation Directly
- Many functions return lists
- Empty list
  - `myList[]`
  - `myList = list()`
Lists - Continuation

- Sorting, Appending, Inserting

```python
>>> myList = ['Stefan', 'Hubert', 'Katharina', 'Maria', 'Monika']
>>> dir(myList)
['append', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
>>> myList.sort()
>>> myList
['Hermann', 'Hubert', 'Katharina', 'Maria', 'Monika', 'Stefan']
>>> myList.reverse()
>>> myList
['Stefan', 'Monika', 'Maria', 'Katharina', 'Hubert']
>>> myList.append('Theresia')
>>> myList.insert(3, 5835)
>>> myList
['Stefan', 'Monika', 'Maria', 5835, 'Katharina', 'Hubert', 'Theresia']
```
### Pythonlists – More methods

- **Deleting, Overwriting, Counting**

```python
>>> myList.append("Stefan")
>>> myList
['Stefan', 'Monika', 'Maria', 5835, 'Katharina', 'Hubert', 'Theresia', 'Stefan']
>>> myList.count("Stefan")
2
>>> myList.count("Monika")
1
>>> myList.remove("Stefan")
>>> myList
['Monika', 'Maria', 5835, 'Katharina', 'Hubert', 'Theresia', 'Stefan']
>>> myList[2] = "Matthias"
>>> myList
['Monika', 'Maria', 'Matthias', 'Katharina', 'Hubert', 'Theresia', 'Stefan']
```
Dictionaries

- Unsorted Lists with key:value pairs
- Creation
  - `myDict = { "name" : "Tino" }`
- Used as lookup table
- Keys have to be unique
  > Overwrite value by using the same key
- Recursive, values can also be dictionaries
Dictionaries - Continuation

- Access keys and values separately
- Delete values: del dic[key]

```python
>>> myDic = {'Mother':'Maria','Father':'Hermann','Son':'Hubert'}
>>> dir(myDic)
['clear', 'copy', 'get', 'has_key', 'items', 'keys', 'popitem', 'setdefault', 'update', 'values']

>>> myDic.keys()
['Mother', 'Son', 'Father']
>>> myDic.values()
['Maria', 'Hubert', 'Hermann']
>>> myDic.items()
[('Mutter', 'Maria'), ('Sohn', 'Hubert'), ('Vater', 'Hermann')]
```
Casting (Changing the var type)

- Sometimes you want to explicitly change a variable
- Use var type name
  - e.g. int(), float(), str(), list()

```python
>>> name = "Marko"
listName = list(name)
>>> listName
['M', 'a', 'r', 'k', 'o']

>>> a = "2"
>>> b = "8"

>>> sum = a+b
>>> print sum
28

>>> sum = int(a)+int(b)
>>> print sum
10
```
Exercise 1.3 – 1.6

- Get to know those different variable types
Chapter 2

- Conditions and loops
Course content forenoon

- Chapter 0: Introduction to Python
- Chapter 1: Variables, Lists, Dictionaries
- Chapter 2: Loops and conditions
- Chapter 3: FME Workspaces and Python
Builtins

- Built-in methods
  - `dir()`, `print()`, `range()`, `len()`
  - `type()` shows type of a variable
  - `cmp()` compares two objects
  - `round()` rounding
  - `max()`, `min()` get the minimum or maximum value of a list

→ Overview: `dir(__builtins__)`
## Conditions – if/else

- Do different things
- **Keywords**: `if`, `elif`, `else`
  - `":"` and Intendation define your code logic
- No code for one condition
  - Use keyword `pass`

```python
# Schema
if <Condition>:
    Statement 1
...
elif <Condition>:
    Statement 2
else:
    ...
<Normal codepath>
```
**Comparisons**

- **Standard**: `==, <, >, <=, >=, !=, <>`
- **Operator „is“** compares, if objects are the same
- **Operator „in“**: Does a list contain the element

```python
>>> a = range(0,10)
>>> b = range(0,10)
>>> a == b
True
>>> a is b
False
>>> a = b
>>> a is b
True
>>> a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> 5 in a
True
>>> -5 in a
False
>>> c = 3
>>> c <> 4 # c !=4
True
```
Boolean operators

- "and" => logical and
- "or" => logical or
- "not" => negation

5 > 6
False

5 > 6 or 10 > 2
True

5 > 6 and 10 > 2
False

>>> user = "Maria"
>>> if not user == "Marko":
    print "You are not Marko."
You are not Marko."
For loops

- Repeat some code multiple times
- Iterate over lists
- Again, use "::" and indentation for structure

```python
>>> sum = 0
>>> a = range(1,5)
>>> for j in a:
    sum = sum + j
print sum
1
3
6
10
```
While loop

- Does also repeat some code part
- Repeats as long as the condition is True

```python
>>> transformer = ['Creator', 'Tester', 'Inspector', 'Sampler']
>>> i = 0
>>> while i < len(transformer):
    print(i+1,", transformer:", transformer[i]
    i = i + 1
1 . transformer : Creator
2 . transformer : Tester
3 . transformer : Inspector
4 . transformer : Sampler
```
Exit Loop early

- Keyword: break
- Continues where the loop code ends

```python
a = range(0,100) #0..99
for i in a:
    if i == 50:
        #gefunden
        print "Found 50!"
        break
print "Normal codepath."
```
Keyword: continue

Skips code, continues with next iteration

```python
# Only even numbers
a = range(1, 11)
i = 0
for i in a:
    if i%2 != 0:  # Modulo
        continue
    print i
```

2 4 6 8 10
Help

- Local Python Help
  > Start > Programs > Python 2.7 > Python Manuals

- Internet
  > http://python.org/
  Documentation, PythonWiki ....
Exercise 2

- Write a guess-the-number-game
Chapter 3

- FME Workbench and Python
Course content forenoon

- Chapter 0: Introduction to Python
- Chapter 1: Variables, Lists, Dictionaries
- Chapter 2: Loops and conditions
- Chapter 3: FME Workspaces and Python
- Import modules with **import**
- Some variants
  - `import fmeobjects`
    - `myFeature = fmeobjects.FMEFeature()`
  - `from fmeobjects import FMEFeature`
    - `myFeature = FMEFeature()`
  - `from fmeobjects import FMEFeature as feat`
    - `myFeature = feat()`
Python functions

- Keyword: `def`
- Need a name
- Parameters in round brackets

```python
def processFeature(feature):
    feature.setAttribute("name","Tino")
```
class FeatureProcessor(object):
    def __init__(self):
        self.constant = "Tino"
    def input(self, feature):
        feature.setAttribute("Name", self.constant)
        self.pyoutput(feature)
    def close(self):
        pass
FMEObjects Documentation

- `<FMEHOME>/fmeobjects/python/apidoc/index.html`

- Have a look especially at
  - FMELogFile
  - FMEFeature
FMELogFile()

- Much better than doing print("My message")
- Gets output to the Workbench Log window and the default logfile
- Use different severity levels:
  - fmeobjects.FME_INFORM
  - fmeobjects.FME_WARNING
  - ...
- Hint: Activate "Log timestamp information" in the FME Runtime options
Exercise 3

- Add a Startup script
- Use a function and a class implementation in the PythonCaller
Chapter 4

- Configuring Eclipse + PyDev
Course content afternoon

- Chapter 4: Configuring Eclipse
- Chapter 5: Python WorkspaceRunner
- Chapter 6: Custom Format Reader
- Chapter 7: Debugging
Eclipse

- Very extensive programming IDE
- Python Plugin PyDev
- CVS/SVN/GIT Support

- On startup choose `C:\PythonTraining\eclipse_workspace` as workspace directory
Configuring Python Interpreter
Configure FMEObjects

- To get code assist, the FMEObject module has to be configured as Forced Builtin
Exercise 4

- Configure Eclipse
- Run the TestFME Project
Chapter 5

- FMEWorkspaceRunner
Course content afternoon

- Chapter 4: Configuring Eclipse
- Chapter 5: Python WorkspaceRunner
- Chapter 6: Custom Format Reader
- Chapter 7: Debugging
FMEWorkspaceRunner

- Helps you to run FME workspaces from your Python code
- Automation
- Batch Processing
- More flexible than Batchscript
- Much less code than using
  - `os` module
  - `subprocess` module
Use the Python WorkspaceRunner interface to retrieve information from workspaces:

- `getPublishedParamNames(workspace)`
- `getParamValues(workspace, paramName)`
- `getParamDefaultValue(workspace, paramName)`
- `getParamLabel(workspace, paramName)`
Use the Python WorkspaceRunner interface to run your FME workspaces:

- `run(workspace)`
- `withParameters(workspace, parameters)`
- `promptRun(workspace)`
Exercise 5

- Open the DemoWorkspaceRunner project in Eclipse
- Your goal is to complete the python code to get a successful run of the workspace
Chapter 6

- Custom Format Reading
Course content afternoon

- Chapter 4: Configuring Eclipse
- Chapter 5: Python WorkspaceRunner
- **Chapter 6: Custom Format Reader**
- Chapter 7: Debugging
Custom Format Schema

FEATURE
ID: 638775314
AMENITY: school
NAME: Lord Byng High School
-123.1929566
49.2588828

Begin of a new feature
Attribute: Value
Attribute: Value
Attribute: Value
Geometry x
Geometry y
PythonCaller

- Each text line enters as one feature
- For each line you have to decide (if...):
  - It is a "new Feature" line
  - Contains an attribute:attributevalue pair
  - Contains the x or y coordinate

- Write code in CustomReader.py
- Use this file with PythonCaller (Entrypoint)
Creating a FMEGeometry

- Create a Feature:
  ```python
  myFeature = fmeobjects.FMEFeature()
  ```
- Create a Geometry:
  ```python
  geom = fmeobjects.FMEPoint(x, y)
  ```
- Attach Geometry to Feature:
  ```python
  myFeature.setGeometry(geom)
  ```
Exercise 6

- Open the CustomReader project in Eclipse
- Open the contained workspace with FME

- Please replace the code in CustomReader.py with

http://goo.gl/5hb3gx
Chapter 7

- Debugging
Course content afternoon

- Chapter 4: Configuring Eclipse
- Chapter 5: Python WorkspaceRunner
- Chapter 6: Custom Format Reader
- Chapter 7: Debugging
Debugging

- Debugging Python scripts directly in FME:
  - Not possible
  - But you can use an external debugger

- PyDev includes a remote debugger
Steps to debug

1. Import the module
   ```python
   import sys
   sys.path.append(r"C:\bin\eclipse\plugins\org.python.pydev_3.5.0.201405201709\pysrc")
   import pydevd
   ```
2. Call the debugger in your python code
   ```python
   import pydevd
   ```
3. Start the remote debugger
4. Start FME with your python scrip
Thank You!

- Questions?

- For more information:
  - Tino Miegel, t.miegel@conterra.de
    - @Tmiegel

- For a certificate of participation mail train@safe.com with the course name

- Have a safe trip home!
Mission Control 4 FME Server

- [http://mc4fme.com](http://mc4fme.com) => G+ Beta test group