Python Programming

Beginners Workshop

held at the Fablab Lisbon

January 2016

Requirements for the Workshop

- Installation of the Python program
  Download from http://www.python.org the current version of python3
- At http://python-rocks.blogspot.com there is a growing number of articles, that give additional explanations and useful links for this workshop.

The installation of Python provides us with a simple editor and a console window to execute Python commands and programs:

IDLE

There are better editors and development tools available, but for the beginning, IDLE should work well.

„I need more information!“

Many sources of help and useful information are available:

- The original Python documentation (F1 in the IDLE)
  ➡ create a link to the local documentation in your working environment
- Python documentation online at https://docs.python.org/3/
- There are many online tutorials on the web,
  e.g.: http://www.python-course.eu/python3_course.php
  or, more advanced: http://docs.python-guide.org (hitchhiker's guide)
- Documentation and tutorials are also available in portuguese:
  https://wiki.python.org/moin/PortugueseLanguage
- Ask your tutor (by mail: python-ws@bodens.de)

➡ **Go and search the Internet**

Usage of the Python shell (console)

Use of literal values

- Enter numbers, like: 3, 17 or 23482398473982457293487592465
- Enter simple calculations: like 2+3 or 7*12
- Try divisions, like: 12/4, 9/3, 11/5
- See the difference between integers and floating point numbers (3.14)
- Enter text strings, like: „hallo world“ or 'I am tired'
- Try out different mathematical operators: +, -, *, **, /, //, %, &, |
- Try the logical operators: <, >, ==, !=

➡ **The Python shell can be used as a calculator**

More usage of the Python shell

Use of variables

- Try to assign a name to a value (variable), like: a=7, b=2*3.9, t="what"
- Enter the name of the variable
- Understand the difference between a name and a (literal) value

Call of functions

- print()
- len() - length of strings
- int() - convert a value to an integer, like: int("765"), int(3.75)
- max(), min() - for numeric values
- input() - what does happen here?

Writing a first program

The infamous „Hello World“

- Use the IDLE editor (or any editor of your choice)
- Write a print statement
- Save the program text as a .py file
- Execute it (in the IDLE shell or on the command line)

The second program: Interaction with the user

- Ask the user for his/her name
- Write a greeting message like: „Hello Alice, it's nice to meet you“
- Save and execute the script

Control the flow

Conditional execution of a statement

- Let the user enter a number,
- then tell him/her, if the number is even or odd
- Use an if … else … statement

Indentation matters (no braces! - but colons!)

- Execute a print statement several times
- Use a for loop and the range function
- Save and execute the script

Iterations

„for item in list:“

- Loop over an iteration
- Use „break“ to leave the loop
- Use „continue“ to skip the rest of the block
- „for“ also has an optional „else“ part

„while condition == True:“

- Loop with a condition expression
- Infinite loops can be useful
- Usage of „break“ and „continue“ as in „for“ loops

Import of Modules

Import of modules from the Python standard library

- Math-Module
  - import math - offers these functions (examples):
    - math.sqrt() calculates the square root
    - math.sin(math.radians(30)) => 0.5
- Time-Module
  - import time
    - time.time() - returns the current time in seconds (as a float)
    - time.sleep(2.0) - waits for 2 seconds
- Random-Module
  - import random
    - x = random.random() - returns a float: 0.0 <= x < 1.0
    - x = random.choice(list-of-items) – returns one of the items in the list

Workshop Projects (1)

Things we could do in the next few weeks

- Sure not all of them – the time is too short
- Perhaps not each of them – some may be too complex

- Easy Projects:
  - Rock, Scissor, Paper (simple game against the computer)
  - Hangman (word guessing)
  - Mastermind (decode numbers)
  - Towers of Hanoi (a puzzle)

Workshop Projects (2)

Things we could do in the next few weeks

- Useful Projects
  - Work with files and directories, search and rename Files
  - Music player
  - Read and write Office documents (Excel), process data
  - WebScraper – Get data from the Internet
  - ScreenBot – Automate browser games (not only for games)
  - Gui-Application programming

- Simulations
  - Game of Life
  - Elevator Simulation

Workshop Projects (3)

Things we could do in the next few weeks

- Artistic and Creative
  - Maze generator
  - Fractal graphics
  - Fractal Gallery (work with HTML)
  - Turtle graphics

- Other
  - Work with the Laser Cutter (SVG)
  - Work with the Raspberry Pi
  - Client-Server-App (Group Chat application)
  - Text-Adventure game
  - Puzzle solving

Rock, Paper, Scissor

Write a Program to play RPS with the computer

- The rules are simple:
  - Rock wins against Scissor (makes it 'unsharp')
  - Scissor wins against Paper (cuts it)
  - Paper wins against Rock (wraps it)
  - Equal bets are draws, no win points
- The computer makes an internal bet on R, P or S
- The user enters his bet
- The computer shows the result. (my bet, your bet, I win, you win, draw)
- The computer counts the win points.
- Who first gets 10 win points, wins the game

*==> A **detailed step-by-step description** of a possible solution is now **online.***

Even - Odd

Write a Program to distinguish even and odd numbers

- The program asks the user for a number
- It determins if the number is eve or odd
- A message is written about the result
- The user is asked again, until he/she wants to quit

This is easier than the RPS project, but it shall be written, using functions

- Use a function for the main part of the program
- Use a function for the user input
    - This function could also check, if the user input is „valid“
- Use a function for the even-odd checking

OOTS Scripts

OOTS Documents - Learn Python by example

Each OOTS document tries to cover one important concept of Python.

Each page of an OOTS document should be understandable by itself.

The OOTS documents are pesented during the workshop and are available for download from the workshop homepage.

The input text and a „text-version“ of the output text file are also available for examination.

Its **an exercise**, to look at the statements and **determine the output**

If the output of a statement is not completely clear: **ASK**!

OOTS Scripts

Out Of The Shell - Show Python at work

- OOTS scripts are lists of python expressions and statements
- The scripts are converted into PDF documents
- Each page shows a number of „expressions“ and „statements“
    - '#' (green) are comments. Comments try to explain, what's going on
    - '>>>' (black) are the expressions or statements
    - '. . .' (black) is for the continuation of a multi-line statement
    - '==>' (blue) are results or return values
    - 'p( )' (violet) shows printed output
    - 'err' (red) displays error descriptions

OOTS Scripts

Out Of The Shell - Show Python at work

- The black lines are „expressions“ or „statements“
    - The are „evaluated“ or „executed“ like in a Python shell
    - The difference is either obvious or not important. It will become clear over time
    - executed statements (like assignments) have no return value
    - evaluated expressions return a value, but it may be None
    - both can have side effects (like printing some text, defining a name, ...)

Exercise: Basic_Gui

Using a simplified Graphical User Interface

The first sample projects were executed on the command line. Program output is written to the screen, User input is entered on the command line, followed by more program output. Its easy to understand and to code, but not exactly beautiful.

To learn about GUI-Programming, the first step is to use a simplified GUI interface. This is still text-oriented, but allows us to understand the GUI control flow, which is different from the 'normal' control flow.

Normal control flow means, that we know where the program starts and we can follow the execution of statements step-by-step until the program ends

In a GUI application this is different. The program initializes the GUI and then gives up control.

Exercise: Basic_Gui

The magical 'event-loop'

A GUI tool is a complex piece of software, which does most of it's work in the background. After the GUI gets control, it paints the visible parts of the application to the screen and then enters the event loop.

The event-loop is the part of the GUI tool, that reacts to user actions. Most of the user-actions (=events) are handled by the GUI autonomously. The GUI handles resizing of the window, scrolling of text, opening and closing menus, and much more, even the termination of the of the application. The application program (our code) does not know anything about what happens on the screen.

Only in very special cases, the event loop may decide to give up control for a short moment, by calling a callback function in our application code. The callback functions should react on the data, which is eventually delivered, and give back control as soon as possible (return from the callback).

Exercise: Basic_Gui

GUI design and event-handlers

Writing an App with a graphical user interface requires many steps:

- Design the elements and the layout of UI elements like menues, buttons, entry fields, data tables, images, colours, scroll-bars, ...
- Setup of the GUI toolkit, trying to get our design expressed in the language of the toolkit (which may be incredibly hard).
- Write all the code, which should handle some or many user actions.
- The code we write will also try to update the elements of the GUI
- Connecting our pieces of program code to some elements of the GUI in the form of 'callback functions'.
- Also connect our code to other elements of the GUI to achieve the desired changes is the visible parts of the GUI.

Exercise: Basic_Gui

For learning purposes: Use a prepared GUI-Application

Download the basic_gui.py from the github scripts folder (it has changed)

Put that file in the same folder, which will contain you application.

Open a new python file and start to edit it:

```
# python 3
import basic_gui
```

Start a class definition, which we need to save some global variables

```
class G():
    appgui = None
    # more variables can be added here later
```

The class G is used as a global container for variables we will need for our appliction

Exercise: Basic_Gui

Create a CmdlAppGui Object

Define our main function, which will start the GUI application

```
def main():
    G.appgui = basic_gui.CmdlAppGui("Title Text", text_input)
    G.appgui.start()
```

We save the appgui object as a global variable in G (as an 'attribute' of G)

The G.xxxx format is called 'attribute access'. We can create new attributes and access existing attributes at any time.

The name 'text_input' refers to a function, which we will define next. This will be used as 'callback function'.

Exercise: Basic_Gui

Define the callback function

Define this:

```
def text_input(text):
    print("received input: '{}'".format(text))
    G.appgui.put_msg("> {}\n".format(text), style=0)
    G.appgui.put_data("Data: '{}'\n".format(text), style=1)
    if text == 'exit':
        G.appgui.finish()
```

This is a very simple version of a callback.

It shows, which methods are available on the appgui object:

- receive the text, which the user entred in the input field
- send a text to the left window: put_msg()
- send a text to the right window: put_data()
- call the finish() method to end the application

Exercise: Basic_Gui

Extend 'text_input()' to build your own application

Proposal: build the RPS game in the GUI style. Make a smart use of the 2 windows.

Where is the benefit?

Understanding the callback mechanism is very useful for a number of possible applications. All graphical user interfaces use this technique.

Webserver applications work on the basis of callbacks.

As soon as some hardware is involved, like in micro-controllers, this callback mechanism is probably used. Same is true for many simulations.

Last, not least: Its fun!

Access to Files

„open() for reading“

- Open returns a „file“ object
- Beware: strings are not bytes. Encoding matters
- A file can be used as an iterator over the lines
- Closing files „.close()“ is recommended, can be manually, or automatically: „with“ context

Objects and Classes

After the Cookie Fest

- Explain the concepts of Python as dry facts is second best.
- The oots-format of description should be more interesting, as it invites for experimenting
- The basic ideas and the syntax of objects and classes are available in the oots documents
- A few 'tricks' in the oots are there to expose, how classes and objects are connected and what really happens, when a new objects is created
- In a 'normal' program there should be no surprises for the person who needs to read and understand the code.
- There may be reasons for 'un-normal' programs, but they rather be good reaons.

Objects and Classes

How to use objects and classes

Objects are a combination of methods and data.

- Objects represent some 'noun', something that 'is' and 'has', and can be described in a reasonable number of words and sentences. If it is hard to describe, it is probably not a good example for an object.
- Methods of an object describe what the object 'does'
- Names of objects and methods can be used in meaningful sentences
- Objects have preferably one responsablity.

Objects and Classes

How to use objects and classes

Classes define the features of an object

- The __init__() method can be considered a part of the class.
- The initialization process constitutes the identity of the object
- It is called internally only once at the creation time.
- After the initialization
  - an object is consistent,
  - all methods show a defined behaviour

Objects and Classes

The Account project

This project is about (classical) batch processing. The program is started, gets its data from some input source, processes the data, and writes the result to some output device

- The program uses files, which are stored in some data folder.
- The internal processing uses objects from an Account class
  - The Account class in the oots examples is a startig point
- Eventually there are other classes, like Client or Transaction
  - As soon as an object gets 'too big', it better is decomposed
- The output is a sequential file, which is nicely formatted for printing

„But this should be a database application!“ - Yes, soon, perhaps

The Account Project

File access

The task starts with reading a file.

- on github => scripts there is the file_io_sample.py, which shows the basic syntax of how to read and write a file. More information can be found in the Python documentation under
    - Python Tutorial => Input and Output => Reading and Writing Files
    - Standard Library => Builtin Functions => open()

  or under
    - python-ws@bodens.de
- What may appear difficult: interpreting (=parsing) the input data lines
    - use the .split() method, with its varius options
    - see: Standard Library => Builtin types => Text type => String Methods

The Account Project

Steps for a Solution

There are two input files on github => resources:

- setA-clients.txt   => this file defines the clients for our Account app
- setA_transactions => these are the transaction, that feed our accounts

The first step will be to read these files and get the content

The content of the clients.txt will define objects of a Client class and objects of an Account class. There is more than one client and there is more than one account per client.

The content of the transactions.txt will be used to start deposit and withdraw actions on the account objects.

The account objects need an internal transaction history

Finally a result file account-transactions.txt shall be written.

The Account Project

Output from the Account Process

The output file shall contain one page per account.

=> `'\f'` is for 'form feed' – this starts a new page on the printer

The ouput is ordered by client names and by account numbers.

Each output account page starts with the name of the client.

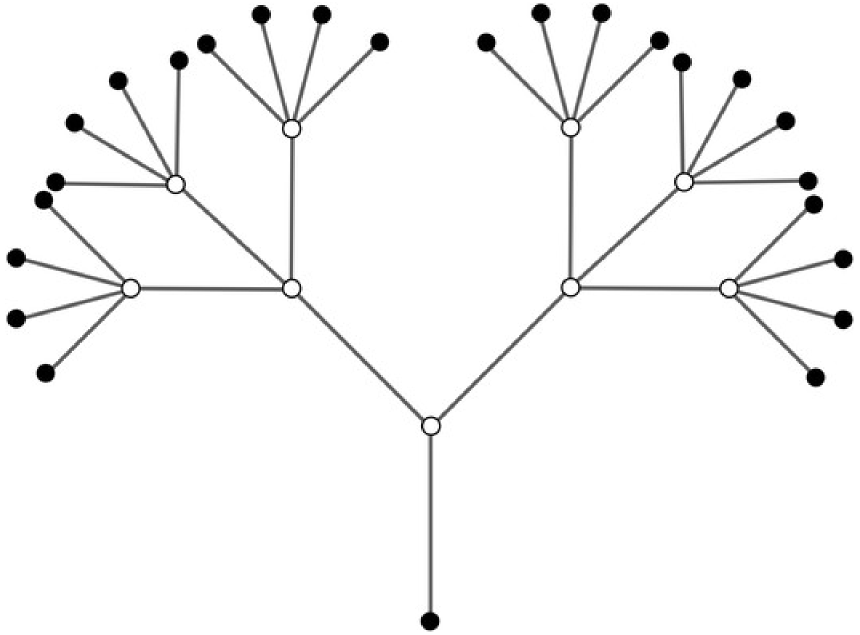
then there is the initial account balance

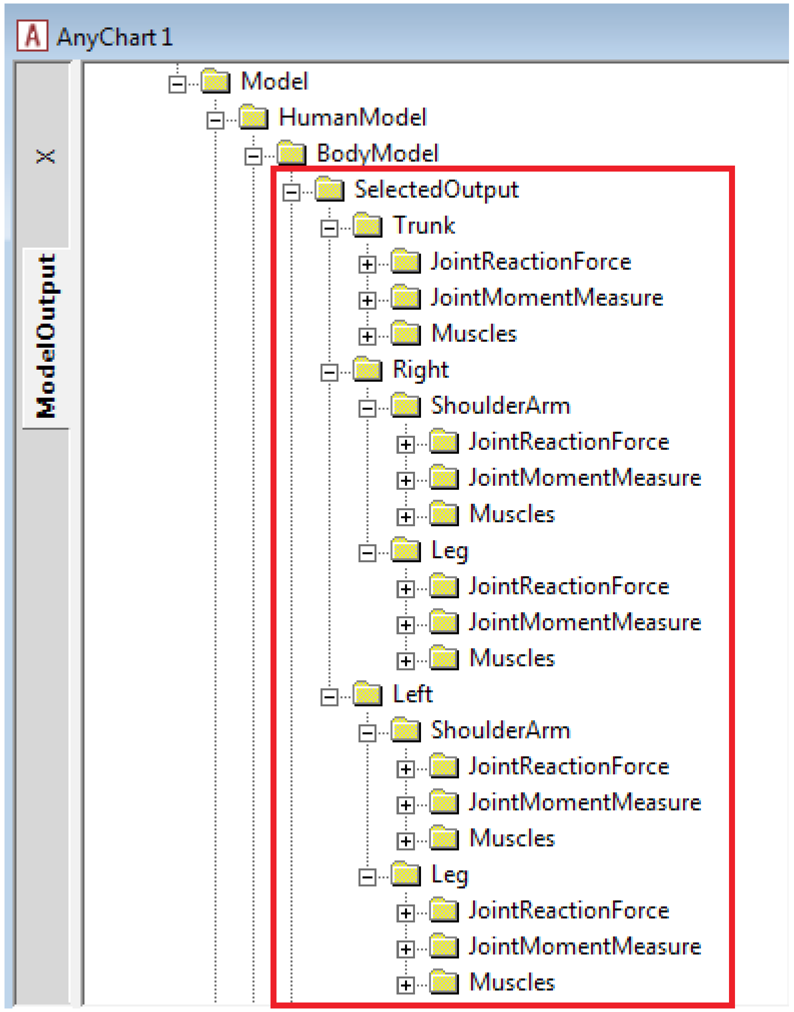
the all transactions follow in chronological order

At the end there is a line giving the final account balance

The example in the oots-ObjectsAndClasses.pdf should work as a guide

Recursion





```
Model
  HumanModel
    BodyModel
      SelectedOutput
        Trunk
          JointReactionForce
          JointMomentMeasure
          Muscles
        Right
          ShoulderArm
            JointReactionForce
            JointMomentMeasure
            Muscles
          Leg
            JointReactionForce
            JointMomentMeasure
            Muscles
        Left
          ShoulderArm
            JointReactionForce
            JointMomentMeasure
            Muscles
          Leg
            JointReactionForce
            JointMomentMeasure
            Muscles
```

Recursion

How to explain Recursion?

„Before we can talk about 'recursion',
we must understand, what 'recursion' is“

- The phrase is quoted often as an example of 'recursiveness'

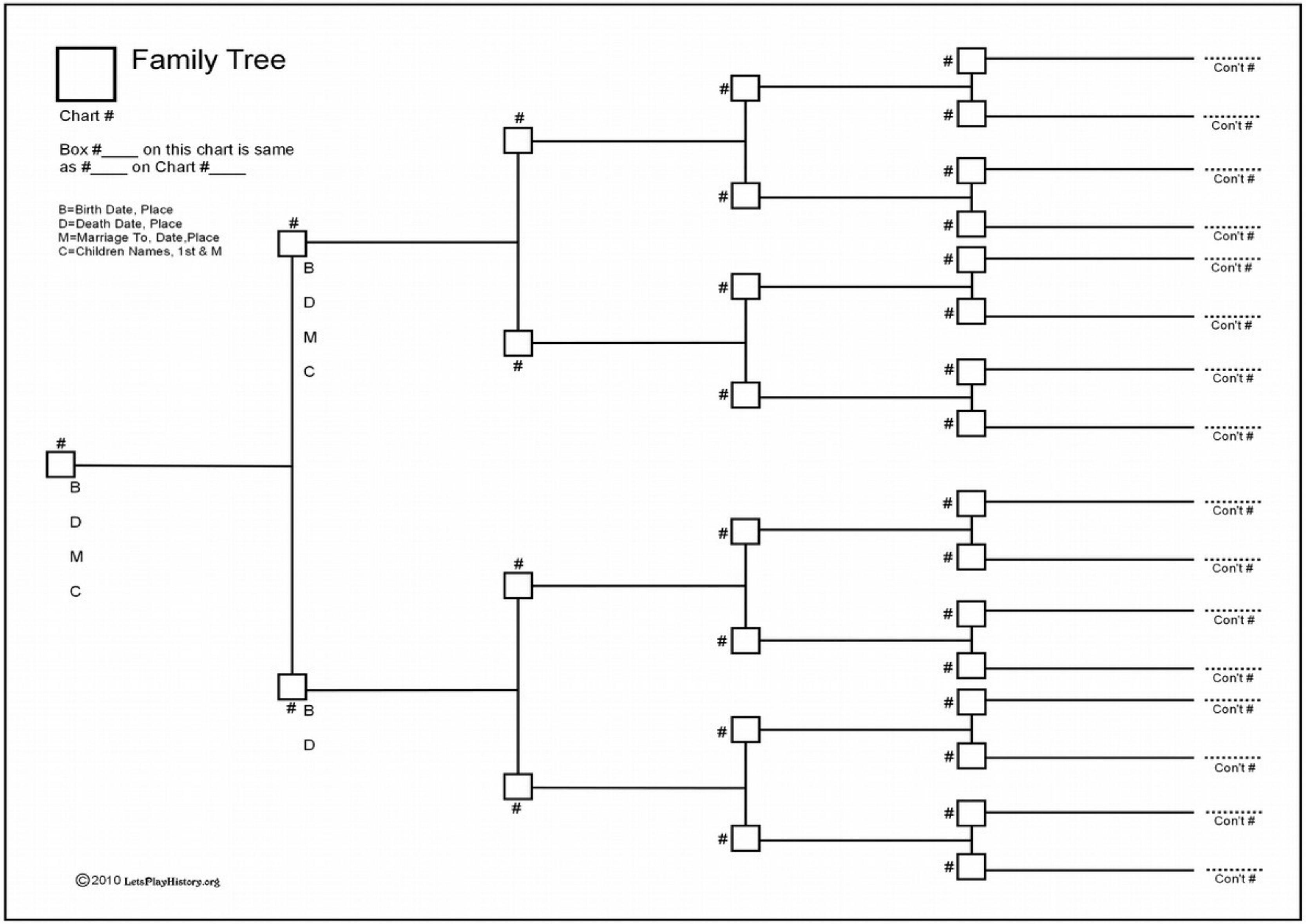
The real world is full of recursive phenomena

Because of this programmers often have to deal with recursive data

Let's do some 'Genealogy', that is studies of the ancestry.

Many people start by asking: 'who are the parents of my grandparents'

Genealogy


Family Tree
Chart #
Box #____ on this chart is same as #____ on Chart #____
B=Birth Date, Place
D=Death Date, Place
M=Marriage To, Date,Place
C=Children Names, 1st & M
#
B
D
M
C
Con't #
©2010 LetsPlayHistory.org

Genealogy

To keep it simple: ask about parents (2) not children (0..n)

- make a Person class
- create a person object for yourself or your child
- the identity of the person should be name, and year of birth
- create another two person objects for the father and the mother
- the Person class needs two functions: has_father and has_mother
- the argument is of course a person object

- try to setup 4 generations (use imaginated names and dates)
- feel the difficulty: The data is hierarchical (recursive), but our program code (or paper) is linear

Recursion (again)

Functions for recursive Data structures

Now we have a family tree in form of many interconnected person objects.

Try to write a function, which returns the oldest person in the family tree.

```
def get_oldest(person):
    ''' function returns the oldest known person in the family tree'''
    .... # some magic here
    return person_object
```

XX

XX

XX

- XX
  - XX

XX

XX

XX

- XX
  - XX