Directory Index Building

Verschiedene Verzeichnisdarstellunen: nach Name, Datum, Sprache, ...

engelbert gruber

06.04.2020
2ND GRINDED SCIENCE

INTRODUCTION

Der Inhalt eines Verzeichnis mit Unterverzeichnissen soll auf verschiedene Arten dargestellt werden: sortiert nach Datum (was gibt es neues), Namen, Tags: Programmiersprache, Sprache, ...

HYPOTHESIS

Die geänderte Darstellung ermöglicht es Neues zu finden, ohne es zu suchen.

MATERIALS

1. python3
2. Wahrscheinlich ein webserver, HTML ist ein sehr bequemes Ausgabeformat.

PROCEDURE

1. Step by step

DATA

Wir haben zurzeit diesen Inhalt

```
./ToDo and InWork.html
./2020-04-03/python notes.pdf
./2020-04-03/arduino Serial IO.pdf
./2020-04-03/java notes.pdf
./2020-04-03/java notes.docx
./2020-04-03/arduino Serial IO.docx
./2020-04-03/python notes.docx
./2020-03-18/arduino-oszilloskop-bericht-teil-2.pdf
./2020-03-18/arduino-oszilloskop-bericht.pdf
./2017/S5-HTML-php-SQL-ORM.pdf
./2020-03-17/arduino input.pdf
./2020-03-17/arduino oszilloskop.pdf
./2020-03-24/pwm-reverse-2.pdf
./2020-03-24/pwm-reverse-1.pdf
./2020-03-24/serial-read-1/
./2020-03-24/serial-read-1/serial-read-1.ino
./2020-04-02/Datenbanken.pdf
./2020-04-02/Datenbanken.docx
./2020-04-04/C und C++ mach ich (zur Zeit) nur am
Microcontroller, Arduino, oder als unittest von uC_arduino-Code
am PC.html
./2000/The Test_Code Cycle in XP, Part 1_ Model.pdf
./2000/The Test_Code Cycle in XP, Part 2_ GUI.pdf
```

Die Ablage der Dateien in Tagesverzeichnissen, 2020-04-03, gibt eine klare Zeitzuordnung, sodass beim Kopieren/verschieben nicht auf den Dateisystemzeitstempel aufgepasst werden muss.

Offene Fragen:

- Wieviel Unterverzeichnisebenen durchsuchen wir ?
- Wenn es pdf- und docx-Dateien gibt zeigen wir beide an ?

NOW GO

Unser Programmentwurf mkindexes.py:

```
#!/usr/bin/python3
"""
Make different index views of directory content
"""
```

Die erste Zeile “#!/ …” heisst wir haben ein python3-Programm (Die Zeile ist eigentlich nicht uns sondern, für die unix-shell).

Danach kommt ein python-Kommentar. Jetz GO

```
#!/usr/bin/python3
"""
Make different index views of directory content
"""
# für jedes Verzeichnis
#   alle Dateien mit Verzeichnis in eine Liste schreiben
#   TODO wo bekommen wir Author, Programmiersprache her
#
# Die Liste nach Verzeichnis (=Datum) sortiert ausgeben
#
# Die Liste sortiert nach Dateinamen (=Titel) ausgeben
```

1. Verzeichnis durchsuchen in python

Welches Modul, welche Bibliothek/Library ?

https://docs.python.org/3/library/ das erste Modul in “File and Directory Access” ist pathlib.

Das Handbuch ist … gewöhnungsbedürftig … wenn man Objektorientierteprogrammierung nicht gewöhnt ist, sehr. Aber

Importing the main class:

```
from pathlib import Path
# Listing subdirectories:
p = Path('.')
[x for x in p.iterdir() if x.is_dir()]
# [PosixPath('.hg'), PosixPath('docs'), PosixPath('dist'),
# PosixPath('__pycache__'), PosixPath('build')]
```

Ist dann doch nur zwei, drei Zeilen.

Auf der python3-Kommandozeile:

```
>>> from pathlib import Path
>>> p = Path('.')
>>> [x for x in p.iterdir() if x.is_dir()]
[PosixPath('2020-04-03'), PosixPath('2020-03-18'),
PosixPath('2017'), PosixPath('2020-03-17'),
PosixPath('2020-03-24'), PosixPath('2020-04-02'),
PosixPath('2020-04-04'), PosixPath('2000')]
```

Also die Liste der Verzeichnisse.

Das “[x for …]” heisst `list comprehension` (glaube ich), das zu wissen macht das Leben aber nicht einfacher (noch).

“`for x in p.iterdir()`” ??? `p.iterdir()` muss irgendetwas (eine Liste) zurückgeben, bei dem “for” sich von einem zum Nächsten weiter arbeiten kann.

```
>>> from pathlib import Path
>>> p = Path('.')
>>> p.iterdir()
<generator object Path.iterdir at 0x7f8be225b360>
```

Ein generator object ? iter kommt von iterate/iterieren, Eins nach dem Anderen. Das macht hier mit der for-Schleife irgendwie auch Sinn, denke ich.

Also

```
>>> from pathlib import Path
>>> p = Path('.')
>>> for e in p.iterdir(): print(e)
```

```
..
ToDo and InWork.html
2020-04-06
2020-04-03
2020-03-18
Und so weiter
```

Im Programm mkindexes.py

```
#!/usr/bin/python3
"""
Make different index views of directory content
"""
from pathlib import Path
# für jedes Verzeichnis
p = Path('.')
#   alle Dateien mit Verzeichnis in eine Liste schreiben
#   TODO wo bekommen wir Author, Programmiersprache her
liste = [] # leere Liste
for e in p.iterdir():
    liste.append(e) # an liste anhaengen
#
# Die Liste ausgeben
print(liste)
# Die Liste nach Verzeichnis (=Datum) sortiert ausgeben
#
# Die Liste sortiert nach Dateinamen (=Titel) ausgeben
```

Ausgabe:

```
[PosixPath('ToDo and InWork.html'), PosixPath('2020-04-06'),
PosixPath('.mkindexes.py.swp'), PosixPath('2020-04-03'),
PosixPath('2020-03-18'), PosixPath('2017'), PosixPath('2020-03-17'),
PosixPath('README.html'), PosixPath('2020-03-24'),
PosixPath('mkindexes.py'), PosixPath('2020-04-02'),
PosixPath('2020-04-04'), PosixPath('2000')]
```

Die “[“, “]” am Anfang beziehungsweise am Ende zeigen, dass es eine Liste (array) ist.

Aber hier sind Dateien und Verzeichnisse gemischt und das

PosixPath('.mkindexes.py.swp') will ich sicher nicht sehen, das ist eine temporäre Auslagerungsdatei eines Editors.

Wir haben beim Handbuchbeispiel

```
[x for x in p.iterdir() if x.is_dir()]
```

Gesehen, dass es eine Methode is_dir gibt, das nehmen wir.mkindexes.py

```
#!/usr/bin/python3
"""
Make different index views of directory content
"""
from pathlib import Path
# für jedes Verzeichnis
p = Path('.')
#   alle Dateien mit Verzeichnis in eine Liste schreiben
#   TODO wo bekommen wir Author, Programmiersprache her
liste = [] # leere Liste
for e in p.iterdir():
    if e.is_dir(): # Unterverzeichnisinhalt holen
        for e1 in e.iterdir():
            list.append(e1)
    else:
        liste.append(e) # an liste anhaengen
#
# Die Liste ausgeben
print(liste)
# Die Liste nach Verzeichnis (=Datum) sortiert ausgeben
#
# Die Liste sortiert nach Dateinamen (=Titel) ausgeben
```

Rumms

```
[PosixPath('ToDo and InWork.html'),
PosixPath('2020-04-06/dirindexing.pdf'), PosixPath('.mkindexes.py.swp'),
PosixPath('2020-04-03/python notes.pdf'), PosixPath('2020-04-03/arduino
Serial IO.pdf'), PosixPath('2020-04-03/java notes.pdf'),
PosixPath('2020-04-03/java notes.docx'), PosixPath('2020-04-03/arduino
Serial IO.docx'), PosixPath('2020-04-03/python notes.docx'),
PosixPath('2020-03-18/arduino-oszilloskop-bericht-teil-2.pdf'),
```

```
PosixPath('2020-03-18/arduino-o...
```

2. In html ausgeben

```
f  = open("index-1.html", "w")
f.write("<html><body>\n")
f.write("<ul>\n")
for e in liste:
     f.write("<li> %s </li>\n" % e)
f.write("</ul>\n")
f.write("</body></html>\n")
f.close()
```

Schaut im Webbrowser so aus:

- ToDo and InWork.html
- 2020-04-06/dirindexing.pdf
- .mkindexes.py.swp
- 2020-04-03/python notes.pdf
- 2020-04-03/arduino Serial IO.pdf
- 2020-04-03/java notes.pdf
- 2020-04-03/java notes.docx
- 2020-04-03/arduino Serial IO.docx
- 2020-04-03/python notes.docx
- 2020-03-18/arduino-oszilloskop-bericht-teil-2.pdf
- 2020-03-18/ard

Todo: Sortieren, Hyperlinks, Verzeichnis als Datum

Dateinamen ohne Verzeichnispfad

Im Handbuch steht irgendwo (das Ding ist leider groß) PurePath.name. PurePath ist die Grundlage von Path. Wir probieren

```
>>> from pathlib import Path
>>> p = Path(“.”)
>>> p
PosixPath('.')
>>> p.name
```

```
''
>>> for e in p.iterdir(): print(e.name)
…
ToDo and InWork.html
2020-04-06
.mkindexes.py.swp
2020-04-03

# und so weiter
p = Path('2000')
>>> for e in p.iterdir(): print(e.name)
…
The Test_Code Cycle in XP, Part 1_ Model.pdf
The Test_Code Cycle in XP, Part 2_ GUI.pdf
```

Das wars html neu:

```
f = open("index-1.html", "w")
f.write("<html><body>\n")
f.write("<ul>\n")
for e in liste:
     f.write("<li><a href=\"%s\">%s</a></li>\n" % (e, e.name) )
f.write("</ul>\n")
f.write("</body></html>\n")
f.close()
```

Dann werden nur die Dateinamen ausgegeben, mann kann aber draufklicken und bekommt das Dokument.

Wir machen etwas mehr python3. Es gibt etwas das sich Context nennt (glaube ich) dann kann man sich das f.close sparen.

```
with open("index-1.html", "w") as f:
    f.write("<html><body>\n")
    f.write("<ul>\n")
    for e in liste:
        f.write("<li><a href=\"%s\">%s</a></li>\n" % (e,
e.name))
    f.write("</ul>\n")
    f.write("</body></html>\n")
```

Das ist nicht viel, brauchen tut man es eigentlich dann, wenn irgendetwas schief geht innerhalb des with-Ausdruckes, weil dann f.close automatisch aufgerufen wird.

Sortieren

Das ganze Programm schaut zur Zeit so aus

```
#!/usr/bin/python3
"""
Make different index views of directory content
"""
from pathlib import Path
# startverzeichnis "."
p = Path(".")
# alle Eintraege im Verzeichnis und in den Unterverzeichnissen
liste = []
for e in p.iterdir():
    if e.is_dir(): # Unterverzeichnisinhalt holen
        for e1 in e.iterdir():
            liste.append(e1)
    else:
        liste.append(e) # an liste anhaengen
# TODO woher Author, Programmiersprache, etc

# Ausgabe html: sortiert nach Dateiname
with open("index-1.html", "w") as f:
    f.write("<html><body>\n")
    f.write("<ul>\n")
    for e in liste:
        f.write("<li><a href=\"%s\">%s</a></li>\n" % (e,
e.name))
    f.write("</ul>\n")
    f.write("</body></html>\n")

# Ausgabe html: sortiert nach Datum (das Verzeichnis)

# Ausgabe html: sortiert ...
```

Sortieren müssen wir bevor oder beim Ausgeben:

```
        for e in liste:
            f.write("<li><a href=\"%s\">%s</a></li>\n" % (e,
    e.name))
```

Nicht der erste Treffer von google, aber “Use the source, Luke.” https://docs.python.org/3/howto/sorting.html . Listen haben eine Funktion sort, die die Liste umsortiert.

```
        liste.sort()
        for e in liste:
            f.write("<li><a href=\"%s\">%s</a></li>\n" % (e,
    e.name))
```

Wir bekommen

- The Test_Code Cycle in XP, Part 1_ Model.pdf
- The Test_Code Cycle in XP, Part 2_ GUI.pdf
- S5-HTML-php-SQL-ORM.pdf
- arduino input.pdf
- arduino oszilloskop.pdf
- arduino-oszilloskop-bericht-teil-2.pdf
- arduino-oszilloskop-bericht.pdf

Großbuchstaben kommen vor kleinen. In howto-sorting steht ein Beispiel:

```
        liste.sort(key=str.lower)
        for e in liste:
            f.write("<li><a href=\"%s\">%s</a></li>\n" % (e,
    e.name))
```

Ohhhh error:

```
  liste.sort(key=str.lower)
  TypeError: descriptor 'lower' requires a 'str' object but received
a 'PosixPath'
```

Also nicht ganz so einfach: key= gibt eine Funktion an die einen Path übergeben bekommt oder so. Weiter unten im howto steht etwas wie :

```
    key=lambda student: student.age
```

Wir probieren:

```
        liste.sort(key=lambda p: p.name)
        for e in liste:
            f.write("<li><a href=\"%s\">%s</a></li>\n" % (e,
    e.name))
```

Das ergibt:

- C und C++ mach ich (zur Zeit) nur am Microcontroller, Arduino, oder als unittest von uC_arduino-Code am PC.html
- Datenbanken.docx
- Datenbanken.pdf
- README.html
- S5-HTML-php-SQL-ORM.pdf
- The Test Code Cycle in XP, Part 1   Model.pdf
- The Test Code Cycle in XP, Part 2   GUI.pdf
- ToDo and InWork.html
- arduino Serial IO.docx
- arduino Serial IO.pdf

Jetzt noch einmal lower probieren

```
        liste.sort(key=lambda p: p.name.lower())
        for e in liste:
            f.write("<li><a href=\"%s\">%s</a></li>\n" % (e,
    e.name))
```

Und tatatahhh

- arduino input.pdf
- arduino oszilloskop.pdf
- arduino Serial IO.docx
- arduino Serial IO.pdf
- arduino-oszilloskop-bericht-teil-2.pdf
- arduino-oszilloskop-bericht.pdf
- C und C++ mach ich (zur Zeit) nur am Microcontroller, Arduino, oder als unittest von uC_arduino-Code am PC.html
- Datenbanken.docx

Anders sortieren, nach Datum=Verzeichnis

Der Parameter key= zur Funktion sort bestimmt offensichtlich nach was sortiert wird.

Die “e” in “`for e in liste`” sind Objekte der Klasse PosixPath (am Windows wahrscheinlich WindowsPath) und haben ein Attribut “name”, schauen wir was es sonst noch gibt.

In https://docs.python.org/3/library/pathlib.html steht unter “Accessing individual parts” es gibt name, suffix, parent. Aber beim Umbau von lambda bekommen wir die Fehlermeldung

```
File "mkindexes.py", line 34, in <lambda>
     liste.sort(key=lambda p: p.parent.lower())
AttributeError: 'PosixPath' object has no attribute 'lower'
```

Das bedeutet wir müssen aus dem PosixPath einen String machen

```
# Ausgabe html: sortiert nach Datum (das Verzeichnis)
with open("index-date.html", "w") as f:
    f.write("<html><body>\n")
    f.write("<ul>\n")
    liste.sort(key=lambda p: str(p.parent).lower())
    for e in liste:
        f.write("<li><a href=\"%s\">%s</a></li>\n" % (e,
e.name))
    f.write("</ul>\n")
    f.write("</body></html>\n")
```

Da sind die neuesten unten … aber lassen wir das einmal so.

Überschriften wären schön.

```
liste.sort(key=lambda p: str(p.parent).lower())
prev_e = liste[0] # to know when parent changes
for e in liste:
    if e.parent != prev_e.parent:
        f.write("</ul><div>%s</div><ul>\n" % e.parent)
        prev_e = e
    f.write("<li><a href=\"%s\">%s</a></li>\n" % (e, e.name) )
```

Für die Namenssortierung wären “Buchstaben” hübsch.

```
liste.sort(key=lambda p: p.name.lower())
prev_e = "X” # Erzwinge die Ausgabe des Buchstaben
for e in liste:
    _e = e.name.lower()[0]
    if prev_e != _e:
        f.write("</ul><div>%s</div><ul>\n" % _e)
        prev_e = _e
f.write("<li><a href=\"%s\">%s</a></li>\n" % (e, e.name) )
```

3. Zwischenstand

Wir bekommen zwei index-Dateien einmal sortiert nach Dateinamen, einmal nach Verzeichnis (=Datum).

```
#!/usr/bin/python3
"""
Make different index views of directory content
"""

from pathlib import Path
# startverzeichnis "."
p = Path(".")
# alle Eintraege im Verzeichnis und in den Unterverzeichnissen
liste = []
for e in p.iterdir():
    if e.is_dir(): # Unterverzeichnisinhalt holen
        for e1 in e.iterdir():
            liste.append(e1)
    else:
        liste.append(e) # an liste anhaengen

# TODO woher Author, Programmiersprache, etc

# Ausgabe html: sortiert nach Dateiname
with open("index-name.html", "w") as f:
    f.write("<html><body>\n")
    f.write("<ul>\n")
```

```
    liste.sort(key=lambda p: p.name.lower())
    prev_e = "X" # Erzwinge die Ausgabe des Buchstaben
    for e in liste:
        _e = e.name.lower()[0]
        if prev_e != _e:
            f.write("</ul><div>%s</div><ul>\n" % _e)
            prev_e = _e
        f.write("<li><a href=\"%s\">%s</a></li>\n" % (e,
e.name))
    f.write("</ul>\n")
    f.write("</body></html>\n")

# Ausgabe html: sortiert nach Datum (das Verzeichnis)
with open("index-date.html", "w") as f:
    f.write("<html><body>\n")
    f.write("<ul>\n")
    liste.sort(key=lambda p: str(p.parent).lower())
    prev_e = liste[0] # to know when parent changes
    for e in liste:
        if e.parent != prev_e.parent:
            f.write("</ul><div>%s</div><ul>\n" % e.parent)
            prev_e = e
        f.write("<li><a href=\"%s\">%s</a></li>\n" % (e,
e.name))
    f.write("</ul>\n")
    f.write("</body></html>\n")

# Ausgabe html: sortiert ...
```

Todo: CSS Behübschung, andere Attribute

4. Refactor

Refactoring bedeutet umbauen ohne die Funktion zu ändern, soetwas wie “aufräumen”.

Bessere Namen oder Kommentare und das Entfernen von doppeltem Code.

Doppelt ist bei uns die Ausgabe in eine Datei, aber der Dateiname ändert sich, die

Sortierung, das heisst wir könnten eine Funktion machen die den Namen und die sortierte Liste übergeben bekommt. ABER die Trenner sind einmal parent und einmal der erste Buchstabe des Namen.

```
# Ausgabe html: sortiert nach Dateiname
with open("index-name.html", "w") as f:
    f.write("<html><body>\n")
    f.write("<ul>\n")
    liste.sort(key=lambda p: p.name.lower())
    prev_e = "X" # Erzwinge die Ausgabe des Buchstaben
    for e in liste:
        _e = e.name.lower()[0]
        if prev_e != _e:
            f.write("</ul><div>%s</div><ul>\n" % _e)
            prev_e = _e
        f.write("<li><a href=\"%s\">%s</a></li>\n" % (e,
e.name))
    f.write("</ul>\n")
    f.write("</body></html>\n")

# Ausgabe html: sortiert nach Datum (das Verzeichnis)
with open("index-date.html", "w") as f:
    f.write("<html><body>\n")
    f.write("<ul>\n")
    liste.sort(key=lambda p: str(p.parent).lower())
    prev_e = liste[0] # to know when parent changes
    for e in liste:
        if e.parent != prev_e.parent:
            f.write("</ul><div>%s</div><ul>\n" % e.parent)
            prev_e = e
        f.write("<li><a href=\"%s\">%s</a></li>\n" % (e,
e.name))
    f.write("</ul>\n")
    f.write("</body></html>\n")
```

Dieses lambda-Ding ist eine unbenannte Funktion:

```
lambda p: str(p.parent).lower()
```

Die eine Parameter p übergeben bekommt und von dem das Attribut parent nimmt, dieses in einen String umwandelt und dann in Kleinbuchstaben umwandelt.

Könne wir so etwas auch für die Zwischenüberschriften machen ?

```
# Ausgabe html: sortiert nach Dateiname
liste.sort(key=lambda p: p.name.lower())
prev_e = "X" # Erzwinge die Ausgabe der ersten Ueberschrift
get_subtitle = lambda p: p.name[0].lower()
with open("index-name.html", "w") as f:
    f.write("<html><body>\n")
    f.write("<ul>\n")
    for e in liste:
        _e = get_subtitle(e)
        if prev_e != _e:
            f.write("</ul><div>%s</div><ul>\n" % _e)
            prev_e = _e
        f.write("<li><a href=\"%s\">%s</a></li>\n" % (e,
e.name))
    f.write("</ul>\n")
    f.write("</body></html>\n")
```

Alles im “`with`”-Context kann dann auch für index-date verwendet werden.

```
# Ausgabe html: sortiert nach Datum (das Verzeichnis)
liste.sort(key=lambda p: str(p.parent).lower())
get_subtitle = lambda p: p.parent
prev_e = get_subtitle(liste[0]) # to not print first subtitle
with open("index-date.html", "w") as f:
    f.write("<html><body>\n")
    f.write("<ul>\n")
    for e in liste:
        _e = get_subtitle(e)
        if prev_e != _e:
            f.write("</ul><div>%s</div><ul>\n" % _e)
            prev_e = e
        f.write("<li><a href=\"%s\">%s</a></li>\n" % (e,
```

```
        e.name))
            f.write("</ul>\n")
            f.write("</body></html>\n")
```

Wir machen aus diesem zweimal vorkommenden Code eine Funktion.

Und entscheiden uns auch die erste Überschrift immer auszugeben und setzen prev_e innerhalb der Funktion auf None.

```
# Ausgabe html:
def wr_html_file(filename, liste, get_subtitle):
    prev_e = None
    with open(filename, "w") as f:
        f.write("<html><body>\n")
        f.write("<ul>\n")
        for e in liste:
            _e = get_subtitle(e)
            if prev_e != _e:
                f.write("</ul><div>%s</div><ul>\n" % _e)
                prev_e = _e
            f.write("<li><a href=\"%s\">%s</a></li>\n" % (
                                        e, e.name) )
        f.write("</ul>\n")
        f.write("</body></html>\n")

# sortiert nach Dateiname
liste.sort(key=lambda p: p.name.lower())
wr_html_file("index-name.html", liste, lambda p:
p.name[0].lower())

# Ausgabe html: sortiert nach Datum (das Verzeichnis)
liste.sort(key=lambda p: str(p.parent).lower())
wr_html_file("index-date.html", liste, lambda p: p.parent)
```

5. html und CSS

Jetzt können wir die Änderungen der html-Dateien an einer Stelle vornehmen.

```
def wr_html_file(filename, liste, get_subtitle):
```

```
    prev_e = None
    with open(filename, "w") as f:
        f.write("<!DOCTYPE html>\n<html>\n")
        f.write("<body>\n")
        f.write("<ul>\n")
        for e in liste:
            _e = get_subtitle(e)
            if prev_e != _e:
                f.write("</ul><div>%s</div><ul>\n" % _e)
                prev_e = _e
            f.write("<li><a href=\"%s\">%s</a></li>\n" % (
                                        e, e.name) )
        f.write("</ul>\n")
        f.write("</body></html>\n")
```

Wir machen den html-Kopf etwas besser, mit DOCTYPE, encoding Angabe und einer CSS-Datei.

```
   f.write("<!DOCTYPE html>\n<html>\n")
   f.write("<head>\n")
   f.write("  <meta charset=\"utf-8\">\n")
   f.write("  <link rel=\"stylesheet\" type=\"text/css\"
href=\"style.css\">\n")
   f.write("</head>\n")
```

Und wir geben beim Trenner/Subtitle eine CSS-Klasse an.

```
           if prev_e != _e:
                f.write("</ul><div class=\"sep\">%s</div><ul>\n" %
_e)
                prev_e = _e
```

style.css

```
     .sep {
       background-color: lightgray;
     }
     ul {
       margin-top: 2px;
       margin-bottom: 2px;
```

```
    padding-left: 20px;
}
```

REFERENCES

1. Google
2. stackoverflow
3. https://docs.python.org/3/library/