Scan voor foto's op harde schijf

Using Python

In de loop van de jaren worden veel foto's gemaakt af en toe worden ze van camera/telefoon gehaald en ergens op een harde schrijf van een computer gezet. Daardoor is het overzicht kwijt van wat er aan foto's is.

1. Maak een programma dat een begin krijgt en vanaf daar alle sub-folders scant op zoek naar foto's (voor nu even .jpg, .jpeg en .png)
2. Onthoudt locatie (full-path), grote en datum (locale database of naar JSON schrijven)
3. Zorg ervoor dat het geheel Operating system proof is (op Windows wordt \ als scheidingsteken gebruikt en / op Linux)
4. Vind duplicaten. Kan op naam, maar kan ook op inhoud. Op inhoud is om effectief te houden, bekijk eerst de extensie van het bestand (jpg is anders dan png) de grote van het bestand en als die exact gelijk is ga dan 1 voor 1 alle bytes af of de inhoud ook gelijk is (dat is langzaam dus als de eerste twee voorwaarden niet waar zijn kun je dat overslaan)
5. Sorteer de lijst op datum, en deel ze in (virtuele) folders per jaar in.
6. Maak een Explorer achtig scherm om dit in weer te geven. Inclusief plaatje
7. Geef optie om foto's te veranderen. B.v. te draaien of naar vaste grote om te zetten (handig om dia shows mee te maken).
8. Gebruik AI om bijna gelijke duplicaten te vinden

Scan voor bestanden

Omdat dit via het Operating Sytem gaat is:

```
import os
```

nodig. Met de functie `os.scandir(mijnDirectory)` krijg je dan een lijst terug met gegevens van alle bestanden/folders in de gevraagde folder. Met for kun je er dan doorheen gaan.

```
list = os.scandir('c:/')
for entry in list :
  if entry.is_dir() or entry.is_file():
    print(entry.name)
list.close()  # Sluiten is netjes om systeem resources te laten vrijmaken
```

Een (OS) entry bevat de volgende waardes en functie:

Naam	**Omschrijving**
name	Naam van het gevonden bestand (zonder path)
path	Volledige path van de gevonden entrie
is_dir()	Is deze entrie een directory / folder?
Is_file()	Is deze entrie een echt bestand?
is_symlink()	Is de entry een symbolische link, komt op Linux vaker voor dat je naar één en hetzelfde bestand meerdere verwijzigen hebt. Als dit waar is geeft hij true terug
stat()	Geeft stat_result terug (zie hieronder)

Deel van de stat_result waardes en functies (zie Python docs voor alle)

Naam	**Omschrijving**
st_size	
st_birthtime	Tijd wanneer een bestand gemaakt is uitgedrukt in secondes. Dit attribute is niet altijd beschikbaar, en kan een AttributeError geven. Mocht hij er niet zijn kun je kijken of st_ctime bestaat (eigenlijk niet meer beschikbaar onder windows)

st_mtime	Tijd van de meest recente verandering uitgedrukt in secondes.
st_atime	Tijd dat het bestand voor de laatste keer gelezen is uitgedrukt in secondes.
st_rsize	Echte bestandsgrote op schijf

Datum tijd

Datum en/of tijd zijn altijd een beetje vreemde eend in de bijt van computer talen. Er zijn vele manieren om deze te beware. Het Operating Systeem hier bewaard het aantal secondes sinds 1 jan 1970 je kunt hier een voor Python bruikbare tijd van maken met:

```
d = datetime.utcfromtimestamp(file.stat().st_mtime)
```

En daar kun je dan weer een formatter voor gebruiken om het leesbaar te maken:

```
Print(d.strftime('%d %b %Y'))  # 16 feb 2026
```

Recursieve functies

Stel je hebt de volgende functie:

```
def scanFolders(startFolder)
     print('Inhoud van ' + startFolder)
     list = os.scandir(startFolder)
     for entry in list :
          if entry.is_dir():
               print('[' + entry.name + ']')
               # ***A***
          elif entry.is_file():
               print(entry.name)
     list.close()
```

Dan krijgen we alleen de inhoud van de gegeven directory van de gevraagde folder. Maar willen we ook de onder liggende gaan doen zouden we bij ***A*** een vreemde constructie moeten maken.

```
If entry.is_dir()
```

```
list2 = os.scandir( os.path.join(startFolder, entry.name))
     for entry2 in list2:
          if entry2.is_dir():

               print('[' + entry2.name + ']')
               list3 = os.scandir( entry2.path, entry2.name))

               for entry3 in list3:

               <etc.>

          elif entry.is_file():
               print(entry2.name)

list.close()
```

Os.path.join() voegt de twee namen samen met het juiste scheidingsteken (/ of \ )

Maar je ziet dat dit heel onhandig is met telkens hetzelfde stukje kopieren, zeker omdat je van te voren niet weet hoeveel sub-folders er komen zit je straks bij regel 60.000 entry999 aan het intypen.

Maar we hebben op dat stuk eigenlijk al alles staan wat er nodig is. En het staat ook netjes in een functie die we kunnen aanroepen. Enige nadeel, die functie is hij zelf. Maar dat is dus niet echt een nadeel want dat mag gewoon. Als we ***A*** vervangen door:

```
scanFolders(os.path.join(startFolder, entry.name))
```

Is dan het enige wat nodig is.

Voor je lijsten op beeld zoals het nu doet is het een beetje vreemd op beeld

Inhoud van C:\
[Documents]
Inhoud van C:\Documents
[Test]
Inhoud van C:\Documents\Test
File1.txt
File2.txt
[User]
Inhoud van C:\User
[John]
Inhoud van C:\User\John

FileJohn1.txt
FileRoot1.txt

Waarschijnlijk wil je iets meer in de richting van:

Inhoud van C:\
[Documents]
[User]
FileRoot1.txt

Inhoud van C:\Documents
[Test]

Inhoud van C:\Documents\Test
File1.txt
File2.txt
Inhoud van C:\User
[John]
Inhoud van C:\User\John
FileJohn1.txt

Dit kan dan nog steeds maar maak bij het begin van de functie

```
folderLijst = list()
```

zet bij ***A***
```
folderLijst.Add(Entry)
```

En aan het eind (maar voor de close want die gooit alle entries weg):
```
for entry in folderLijst:
     scanFolders(entry.path, entry.name))
```

Wordt er nog steeds recursief aangeroepen maar op een ander punt in de functie (Voor ophalen alle bestanden is dit niet heel nodig, maar wel netter)

Als laatste kun je dan nog toevoegen dat alleen foto bestanden getoont worden. Het makkelijkst hiervoor is om te kijken of de bestandsnaam eindigd op een geldige extensie

```
If   entry.name.endswidth('.jpg') or
     entry.name.endswidth('.jpeg') or
```

```
    entry.name.endswidth('.png'):
        :
```

En dan alleen die files te tonen / kopiëren

Maak lijst in het geheugen

Het is natuurlijk leuk dat je een lijst op beeld hebt, maar daar kun niets meer mee doen dan naar kijken. Om het werkbaar te maken, moeten we een lijst gaan bijhouden.

De makkelijkste mannier waarschijnlijk is een dict aan te maken met de data erin en dat in een lijst te bewaren.

```
imageList = list()
```

Iets als:

```
BestandData = { “name”: entry.name, “path”: entry.path,
"size”: entry.stat.st_size, "created": entry.stat.
st_birthtime }
```

En dan:

```
imageList.add(BestandData)
```

Als je helemaal klaar bent kun je de lijst dan sorteren met

```
def sortFunc(e):
Return e[‘name’]
imageList.sort(key=sortFunc)
```

Als je wilt sorteren op andere velden is het handig om naar een tekst om te zetten zodat je b.v. op grote + naam kunt sorteren. Maar dan wordt er wel alphabetisch gesorteerd. Voor op grote sorteren kun je b.v.:

```
    def sortFunc(e):
return str(e.size) + e.name
```

Doen. Maar het probeel is dan dat grote 9 byte en naam ‘Alpha’ => ‘9Alpha’ later komt dan 10032 ‘Beta’ => '10032Beta' (deze komt alphabetisch eerder).

Dit kun je oplossen door i.p.v. str() een format te gebruiken:

return “{:10d}”.format(e.size) + e.name

Dit vult het (links) met nullen uit tot totaal 10 cijfers. Dus "0000000009Apha" is nu echt wel voor "'0000010032Beta " alphabetisch.

Op datum: `SortDateTime = e.created.strftime('%Y-%m-%d %H:%M:%S')`

Dit is omdat binnen datum het jaar het grootst is daarna maand en dag. Wordt het b.v.

“2026-02-16 11:51:25” En mocht iets toevallig om hetzelfde moment gemaakt zijn kun je nog + name erbij zetten.

Verder om zaken beter vergelijkbaar te maken zou ik hoofd en kleine letter van een naam negeren door er .lower() achter te zetten dus

```
def sortFunc(e):
     return e.created.strftime('%Y-%m-%d %H:%M:%S')
            + e.name.lower()
```

Kopieer naar maand folders

Je geeft nu de computer een start folder mee, b.v. ‘c:\johnfotos’ en daaronder willen we alle plaatjes gaan onder verdelen.

c:\
    johnfotos

2025

Jan

Apr

Dec

2026

Feb

Jun

Dit betekent dat je dus zelf folder namen moet gaan samenstellen.

Ten eerste uit een datetime kun je de maandnaam halen met:

```
myDateTime.month_name()
```

Dan krijg je alle maand namen in het Engels.

```
myDateTime.month_name(locale = 'French')
```

Geeft de maandnamen in Frans weer.

Maar je kunt hierboven ‘French’ ook door ‘dutch’ (bijna gelijk aan 'nl_NL') vervangen natuurlijk

Alternatief is dat je ergens in het begin van je programma:

```
locale.setlocale(locale.LC_ALL, 'nl_NL') # maakt alles navolgend Nederlands
```

Neerzet. Dan worden zaken als getallen ('.', ',’) en datums in het juiste nederlandse formaat en dag en weeknamen weergegeven.

Jaar krijg je door `myDate.year`

De functie os.path.join(...) combineert alle parameters tot een string met de juiste folder scheidingstekens. Dus voorbeeld

destFolder = os.path.join(baseFolder, str(myDate.year), myDate.month_name()

srcFile = os.path.join(myFile.path, myFile.name)

destFile = os.path.join(destFolder, myFile.name)

Kopieren kan dan met:

```
shutil.copy2(srcFile, destFile) # Vergeet niet shutil te importeren
```

(sh util staat voor shell utility, dus kan hetzelfde als een shell of command-line)

Het probleem komt als je het programma nog een keer uitvoert, dan staat het bestand al in de destination folder en geeft een fout

Als je

```
  if os.path.exists(destFile):

    os.remove(destFile)
```

Ervoor zet wordt het bestand gewist voordat het (opnieuw) gekopieërd wordt

Als laatste als je niet wil kopiëren maar verplaatsen kun je shutil.move() gebruiken.

Ontdubbelen.

Dat kan op verschillende manieren. Ontdubbelen op naam is het eenvoudigst. De lijst sorteren op naam en grote. Als die niet gelijk zijn dan zijn de bestanden niet gelijk. Als ze beide hetzelfde zijn moet er echt op byte niveau gekeken worden of die allemaal gelijk zijn. Als dat het geval is, kan de eerste/laatste van deze weggehaald worden.

Als je niet zeker bent of de naam hetzelfde is moet je alle bestanden van dezelfde grote met elkaar vergelijken.

Vergelijken op byte niveau:

In Python moet je een bestand openen:

```
with open(filename1, "rb") as file1:
     with open(filename2, "rb") as file2:
```

Je kan het ook combineren

```
with open(filename1, "rb") as file1, open(filename2, "rb") as file2:
```

Dan kun je een aantal bytes tegelijkertijd in het geheugen lezen met:

```
buffersize = 1048576
       :
bytes1 = file1.read(buffersize)
bytes2 = file2.read(buffersize)
```

buffersize is een beetje willekeurig (lijkt proefondervindelijk de maximum grote waar dit goed bij werkt (maar maak gerust kleiner om zeker te zijn dat het goed werkt). Als het resterende deel van de file minder dan die hoeveelheid bytes zijn geeft hij alleen maar dat aantal terug. Als her niets meer over is (einde van bestand bereikt) dan is de variabele ‘leeg’. Dus je kunt blijven herhalen en als leeg dan stoppen.

Dan kun je gerust het hele blok met elkaar vergelijken:

```
if bytes1 != bytes2:
     return False
```

Want als de blokken met bytes niet exact gelijk zijn is de rest van de file ook niet hetzelfde dus kun je gelijk stoppen.

Als het einde van het bestand bereikt wordt kun je een true teruggeven:

```
        if not bytes1:
            return True
```

Want als je ervoor al hebt afgevangen dat ze (wel) gelijk zijn hoef je nog maar van 1 te testen of het einde van de file is bereikt.

Wat de truuk een beetje is, zodra iets vind dat niet gelijk is, stop er dan mee en verspil geen (kostbare computer) tijd met de rest.

Als je twee (of meer) dezelfde gevonden hebt, kun je de dubbelen wisselenn of in een aparte lijst onthouden zodat je het aan de gebruiker kunt laten zien en die laten beoordelen.

Visueel programma

Om een windowed programma te maken gaan we gebruik van TKinter.

En om met plaatjes te kunnen werken gebruiken we Pillow (PIL).

Je hebt nodig:

```
from tkinter import *
from PIL import ImageTk, Image
```

Je initialiseert TKinter (TK) met

```
main = Tk()
main.title(“Dit is de window titel, dus wat in de balk bovenin staat")
main.geometry(“800x700”) # geeft breedte en hoogte van het window
```

Daarna gaan we scherm elementen toevoegen. Veel gebruikte elementen zijn

Label – Normaal voor een simpele tekst op beeld tonen zonder interactie
Button – Knop die ingedrukt kan worden
Entry – Standaard input / edit box

Deze kun je aanmaken. Maar dan moeten ze ook nog in het main window komen.

Daar zijn twee ‘methodes’ voor de packer en de grid. Bij de eerste wordt een element op beeld gezet in de volgorde waarin je ze toevoegd. De grote bepaald of het nog achter de vorig kan anders eronder. Je kunt daar parameters aan toevoegen om het links/rechts onder/boven uit te laten lijnen, of zelfs uit te vullen.

```
label = tk.Label(main, text="Geef je naam:", font=("Arial", 10))
label.pack(side=”left”)
```

De andere is een Grid. Hierbij bestaat een beeld uit vakjes als in een tabel of spreadsheet. En per element kun je dan zeggen in welke row en column het element terecht moet komen. Maar je kunt ook zeggen dat een element meerder kolommen of of regels hoog is.

```
label = tk.Label(main, text="Geef je naam:", font=("Arial", 10))
label.grid(row=1, column=0, columnspan=3)
```