Opgave2b Filesystem integrity

Het doel van het programma is de eigenschappen van een filesysteem, zowel de directories en files in kaart te brengen. Deze gegevens moeten worden opgeslagen in een bestand. Deze structuur kan een json-structuur zijn. Een eerste aanzet van zo'n structuur kan zijn:

```
{
    'file.txt': { 'mode', 'lrwxr-xr-x', },
    'slnk.txt': { 'mode', '-rwxr-xr-x', 'slnk': 'file.txt' },
}
```

Boven staande is een voorbeeld. Het is niet volledig en de vraag is wat de beste manier is om een file en zijn naam te representeren. Denk dus goed na hoe je de directory-info en de file-info wilt opslaan. Json is met de json-module gemakkelijk om te zetten naar lijsten en dicts, en is ook nog enigszins leesbaar. Vandaar een gemakkelijk manier om de informatie op te slaan.

Inventarisatie

Het is eerst van belang dat je begrijpt hoe een unix-filesysteem er uit zit. In eerste instantie willen we niet een geheel file-systeem inspecteren, maar slechts een sub tree.

Een directory bestaat uit files en directories (een sort file), maar er zijn ook andere type files. Welke file eigenschappen zijn relevant voor welk type file. Maak een lijst. Bekijk hoe je die eigenschappen met python kan opvragen. Een aantal overwegingen:

- Bedenk hoe je speciale entries in een directory `.` en `..` wilt behandelen.
- Wat is de unieke key van een unix-file?
- Zou je die kunnen gebruiken als key in je datastructuur?
- Hoe kun je een bepaalde file vinden in je data-structuur?
- Hoe kun een van de paden naar de file vinden in je datastructuur?
- Hoe wil je omgaan met symlinks?
- Hoe kun je vastellen dat een file van inhoud gewijzigd is?

Scannen

Zorg dat je met `os.listdir` alle files en directories vindt aan de hand van een opgegeven directory. Let op: als een entry in de directory zelf ook een directory is moet deze recursief aflopen. Zie zie les2a: Roep je scan functie opnieuw aan op een gevonden directory.

Sla deze boom-structuur samen met de eigenschappen van file in je datastructuur op. Bewaar deze structuur, gebruik bijvoorbeeld json.dump. Het is natuurlijk handig als de json-file dezelfde namen als in stat.

Tonen

We willen een overzicht van de gevonden files. Dus druk alle files af met eigenschappen. Graag willen we een en ander automatisch controleren het het format voor het tonen is: fname:,dname:,mode:,... Dus voor elke file een regel.

- file-eigenschappen: tenminste: fname, dname, mode, mtime, ino, uid, gid, size
- file-eigenschappen: extra: nlink, dev (minor/major), slink, chksum.
- de datum afdrukken als yyyymmddhhmmss
- de mode octaal(0o0777) of als rwxrwxrwx.
- het filetype als octaal(0o00) of als: f: file d: directory l: symlink s: socket p: pipe c: char-dev b: blok-dev For ander typen mag een zelf een logische letter bedenken.

De listing voor de voorbeeld-directory (dir met files file1, file2, file3 zit er ongeveer zo uit) We tonen niet alle eigenschappen. Dat is jullie taak.

```
fname:dir,dname:,type:d,mode:rwxr-xr-x,mtime:20200420113520,ino:8662408162,uid:501
,gid:0,size:160
fname:file2,dname:dir,type:f,mode:rw-r--r--,mtime:20200420113359,ino:8662409572,ui
d:501,gid:0,size:1
fname:dir3,dname:dir,type:d,mode:rwxr-xr-x,mtime:20200420113412,ino:8662409575,uid
:501,gid:0,size:64
fname:file1,dname:dir,type:f,mode:rw-r--r--,mtime:20200420113359,ino:8662408188,ui
d:501,gid:0,size:1
```

Hint: De eigenschappen van een file kun je onder Unix/Mac zien met `ls -l` en `stat`.

Vergelijken

Verander wat files in de directory. Scan deze directory opnieuw. Vergelijk beide datastructuren en geeft aan welk files nieuw zijn en welk verwijderd. Geef als de file nog bestaat of er iets veranderd is en zo ja wat.

Rapporteren

Er zijn twee soorten van rapportage (die voor de mens) en die voor systemen. Welke eisen zul je stellen aan een rapportage voor een persoon. En welke eisen zul je stellen aan de rapportage voor een programma. We gaan deze programma's gedeeltelijk automatisch nakijken. Dus we hebben een standaard manier om te rapportages voor programma's.

Elke geconstateerde verandering wordt op een standaard manier beschreven met vijf velden

`<actie>:<fname>:<attr>:<oldVal>:<newVal>`

- actie is A: toegevoegd, D: verwijderd of M: veranderd.
- fname is het pad (path) naar de file, iets als
  /
  /.
- attr: welk attribute (mode, mtime, uid, data, size, ...) is veranderd.
- oldVal: oude waarde
- newVal: nieuwe waarde

Een voorbeeld:

```
M:test/dir2/file21:size:6:10
M:test/dir2/file21:mode:rw-r--r--:rw-r--r--
M:test/dir2/file21:mtime:20200513075527:20200514010000
D:test/file1
A:test/file4
```

HINT: het testprogramma controleert onder ander de volgende velden

```
Standaard: 'fname', 'dname', 'type', 'mode', 'mtime', 'size'
Extra:     'ctime', 'atime', 'ino', 'uid', 'gid', 'slnk', 'hash', 'data'
```

EXTRA: Het is mogelijk te herkennen dat een file van naam veranderd is. Als je dit constateert kun je dit aangeven met:

```
R:<newPath>:<oldPath>
```

Testen

Er is een programma's gemaakt `opg2b_setup.py` Setup creëert een directory met een aantal files en kan een aantal uitvoeren.

Setup heeft drie modes `--clr` ruimt op, `--set` maakt en `--mod` past een aantal files aan.

Run `python3 opg2b_setup.py --set testdir` om een directory `testdir` met testfiles te maken. Na setup moet je je scanner draaien en de inventarisatie bewaren. Vervolgens run je setup met als commando `--mod` (modify) (`python3 opg2b_setup.py --mod opg2b_testdir`). Een aantal files aangepast, verwijderd en aangemaakt. Vervolgens moet je je scanner opnieuw draaien en de wijzigingen vinden.

Setup `--mod` geeft als output een deel van de uitgevoerde veranderingen. Je kunt deze programma's gebruiken om de testen, echter bij het beoordelen van de opgaven worden andere (uitgebreidere) test

gebruikt. Setup geeft ook een voorbeeld van de api-output (de door ':' gescheiden velden)

Opdracht

Je maakt een python3 programma-file `opg2b.py` gebaseerd op `opg2b_skel.py` De eerste aanzet van het programma bevind zich in `opg2b.zip` (net als `opg2b_setup.py`)

Het skeleton programma verwerkt de argumenten en roept de functies aan die je moet implementeren.

je hoeft (in principe) alleen code toe te voegen tussen de commentaar secties:

```
# Student work {{
# Student work }}
```

Het gebruik van `opg2b.py` is als volgt:

```
$ python3 opg2b.py --scan scan.json dir
    # Scan de directory en alle files daaronder
    # Bewaar de opgebouwde structuur in een file.
$ python3 opg2b.py --cmp  scan1.json scan2.json
    # Vergelijk beide files en geef de verschillen aan in api-formaat
$ python3 opg2b.py --show scan.json
    # Print de inhoud van de scan met de show-functie
```

De test-cyclus is ook beschreven in het commentaar van `opg2b_skel.py` en ziet er zo uit:

```
$ python3 opg2b_setup.py --clr dir
$ python3 opg2b_setup.py --set dir
$ python3 opg2b.py --scan scanOld.json dir
$ python3 opg2b.py --show scanOld.json
...
$ python3 opg2b_setup.py --mod dir
M:dir/dir2/file21:size:6:10
M:dir/dir2/file21:mode:rw-r--r--:rw-r--r-T
M:dir/dir2/file21:atime:20210510122730:20200513010000
M:dir/dir2/file21:mtime:20210510122729:20200514010000
D:dir/file1
M:dir/file3:mode:rwxr-x---:rwsr-xr-x
A:dir/file4
$ python3 opg2.py --scan scanNew.json dir
$ python3 opg2.py --show scanNew.json
...
$ python3 opg2.py --cmp scan1.json scan2.json
...
```

De output van het laatste statemnent zou tenminste de regelsvan

`python3 opg2b_setup.py --mod dir` moet bevatten.

Inleveren

Lever `opg2b.py` in op CodeGrade. Een tweetal test-sets zal worden uitgevoerd.

Veel succes. En bij vragen neem contact op met je Docent.