Generelt

En god og solid general struktur for et Python projekt kan se således ud. Det sikrer en overskuelig opdeling af koden og andre vigtige filer.

```
project_name/
├── src/
│   ├── module1.py
│   └── module2.py
├── tests/
│   ├── test_module1.py
│   └── test_module2.py
├── data/
├── .venv/
├── .gitignore
├── requirements.txt
└── README.md
```

Husk at opdatere .gitignore filen for at ignorere filer og mapper det ikke giver mening at committe til github.

Husk at skrive i README.md hvordan projektet skal sættet op efter det er blevet klonet og hvordan det køres.

Husk at inkludere en opdateret requirements fil i projektet.
Den kan laves med:

```
pip freeze > requirements.txt
```

Og bruges til at installere alle dependencies med:

```
pip install -r reqirements.txt
```

PEP 8 er den officielle stilguide for Python-kode, den hjælper med at ensarte stilen for Python kode hvilket gør det nemmere at læse. Den angiver, at indrykning skal være 4 mellemrum pr. niveau, og hvordan indrykningen skal være når lister, dictionaries, og diverse udtryk bliver skrevet på flere linjer. Variabel- og funktionsnavne skal bruge **snake_case**, mens klasser skal navngives med **CapWords**. Der bør være mellemrum omkring operatorer som = i almindelig kode, men ikke når den bruges i parametre eller argumenter. Der skal være 2 blanke linjer før og efter klasser og funktioner, men kun 1 blank linje for funktioner inde i klasser.
https://peps.python.org/pep-0008/
Det er klart nemmere at få IDE'en eller et værktøj til at kontrollere koden automatisk.

Når en fil importeres i Python, så bliver den kørt hvilket inkluderer alle kode linjerne. Man bruger derfor følgende til at sørge for at koden ikke bliver eksekveret når en fil importeres men kun når den bliver kørt.

```python
if __name__ == "__main__":
    main()
```

En tommelfingerregel i forhold software maintainence er at en funktion max bør indeholde 15 eksekverbare linjer. En anden tommelfingerregel er at en funktion bør have en cyclomatic complexity og max nesting depth på max 3-4. Hvis ens funktion overskrider dette bør man overveje om den skal refaktoreres til flere funktioner / mindre komplex kode. Det er selvfølgelig kun tommelfingerregler, så der kan sagtens være konkrete tilfælde hvor en refaktorering ikke giver mening.
Den konkrete udregning af cyclomatic complexity er **ikke** så vigtig.
Det er et udtryk for hvor mange forskellige veje programmet kan køre igennem en funktion, så det vigtigste er bare at huske at if - else, try - except, loops, etc. øger komplexiteten. Men hvis man gerne vil læse mere om den og hvordan den udregnes kan man læse linket.
https://en.wikipedia.org/wiki/Cyclomatic_complexity

Python kan skrives med type forslag for både funktions parametre, retur værdier og variabler. At overtræde disse type forslag vil ikke resultere i en fejl, men afhængigt af IDE'en vil der komme en advarsel om forkert type.

```python
def some_func(param1: str, param2: bool) -> list[str]:
    val: int = 1
```

Python opererer med Truthy og Falsy værdier. F.eks. så vil tomme datastrukturer som lister, sets, dictionaries, strenge osv evaluere som False i if statements. Ligeledes vil disse datastrukturer evaluere som True hvis de indeholder data. Objekter er altid Truthy, medmindre de implementerer `def __bool__():` eller `def __len__():`.
Det betyder at en if statement med en liste kan se således ud.

```python
my_list = []
if my_list:
    print('list have values')
else:
    print('list is empty')
```

Det som der effektiv sker er følgende.

```python
my_list = []
if bool(my_list) is True:
    print('list have values')
else:
    print('list is empty')
```

Vær opmærksom på at selvom en variabel er Truthy, så er den stadig ikke True. Det betyder at statements som test_1 og test_2 begge er False.

```python
my_list = [1, 2, 3]
test_1 = my_list is True
test_2 = my_list == True
```