Python code standard

Indholdsfortegnelse

Code naming convention:

Styles:

Snake case is lower case with underscores:

snake_case

The pascal case is, starting letters are capitalized:

PascalCase

A mix should not be used

Mixed_Case

Convention:

Modules are you python scripts, the individual files that are outside the main.py file, these modules should be places in an orderly folder structure e.g. src/ test/ control/

Module names should be snake case:

python_module.py

when creating classes and methods or function:

class names should be camel case:

PythonClass:

Function, methods and variables should all be snake case:

my_func()

my_var

Constants should be all upper case

How to run python code:

To run a python script, install python in the terminal go to the directory with the python script:

$ Python3 module_name.py

Then python will run the section called if __name__ == "__main__": this is your main function

```
def main():
    print("hello world!")

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

In here I have made a function called main (optional) and that function is the only thing in my if __name__ == "__main__":

Therefore it only runs my main()

How to create classes:

Imports should be as follows:

```
# Standard library imports
import os
import sys

# Third-party imports
import numpy as np
import pandas as pd

# Local application imports
from my_project.module import my_function
```

You don't need the as np or pd, this will make it more condense code

Instead of numpy.array you would write np.array

Where if you want to import all classes and function use:

From module.path import *

For creating a class this is the format, important to notice, all variables in the python class are public and accessible in the entire class when putting self infront.

The __init__ method is the constructor for a python class.

```
class ClassName:
    def __init__(self) -> None:
        self.public_member      # this is comment
        self.__private_member   # Not a true private member
```

Alle methods that are part of the class need to be indented once in relation to the “class ClassName” and they need the self as the first parameter:

```
class ClassName:
    def __init__(self) -> None:
        self.public_member      # this is comment
        self.__private_member   # Not a true private member

    def member_function(self) -> None:
        self.public_member = 1
```

Here a double underscore, can indicate a private member, its not 100% private, it can technically be called outside the class, but pythons replaces it with _ClassName__private_member

Inheritance:

```
class ParentClass:
    def __init__(self) -> None:
        self.public_member = 0       # this is comment
        self.__private_member = ""   # Not a true private member

class ChildClass(ParentClass):
    def __init__(self) -> None:
        super().__init__()          # Call the parent class constructor
        self.__private_member = "Hello"
```

Here we after making the class name we put a parenthesis and inside it the class we would like to inherit from, then in our child constructure we call the parents constructor.

Data classes:

Data classes are classes that specifically hold data, like a struct, it has no methods but it holds variable for us, so its more organized.

```python
from dataclasses import dataclass

@dataclass
class InfoClass:
    version: str
    path = str
    data = str


class ClassName:
    def __init__(self) -> None:
        self.public_member = 0       # this is comment
        self.info = InfoClass

    def set_info(self, version: str, data: str) -> None:
        self.info.version = version
        self.info.data = data
```

Here we use the “dataclass to tell python this class is a data class, important to note, the import is needed. Then we can create instances of that data class.

Good practice:

In python you don’t need to explicitly teel it what the input and output types are, but this is still a good practice, for telling the output type put a -> type are the end of the function declaration.

```python
class ClassName:
    def __init__(self) -> None:
        self.public_member = 0       # this is comment
        self.__private_member = ""   # Not a true private member

    def member_function(self) -> None:
        self.public_member = 1
        print(self.__private_function)

    def __private_function(self, message: str) -> str:
        return "Error: " + message
```

Here we have explicitly told what the parameter message is, it’s a string, and we have explicitly told that it outputs a string, when functions are void (no output), it’s still good to say it has a None output.