# PYTHON PROJECT STRUCTURE

```
APP
├── README.md
├── requirements.txt
├── license.txt
├── data
│   ├── config.json
│   ├── config_test.json
│   └── files
├── APP
│   ├── __init__.py
│   ├── APP.py
│   ├── modules.py
│   ├── exception.py
│   └── test
│       └── app_test.py
```

## ARGPARSE (ARGUMENT PARSING COMMANDS)

```
parser = argparse.ArgumentParser(description='example')

parser.add_argument('--argname', '-a', *)

parser.add_argument('--argname', '-a', *)

parser.add_argument('--argname', '-a', *)

args = parser.parse_args()
```

**\* Flags options**
- [action]    > Action to be taken
- [nargs]     > number of arguments
- [const]     > A constat value required
- [default]   > Value produced if absent
- [type]      > Type to be converted in
- [choices]   > Allowable values
- [requires]  > Optional or not
- [help]      > A brief description of the arg.

## FILES AND FOLDERS

| | |
|---|---|
| README.md | > Description / Installation / Usage / Troubleshooting / Disclaimer / Help wanted / Links |
| requirements.txt | > All external packages. |
| license.txt | > Permissions and copyright. Example: MIT license or X11 |
| config.json | > json file with all the external variables. It works as an interface. |
| config_test.json | > Same json file as before but with testing variables. |
| __init__.py | > Initialize file for the python files. |
| APP.py | > Main APP component, this code calls and manage all modules. |
| modules.py | > Other Python modules needed for the app. |
| exception.py | > Controled errors or exits from the execution. |
| app_test.py | > Paralell file to APP.py but for testing. |

## APP.py

```python
if __name__ == "__main__":

    # Initialize argument parser
    parser = argparse.ArgumentParser()
    parser.add_argument(arg)
    args = parser.parse_args()

    # Declare global variables from parser

    app = App(arg_01, arg_02, arg_03)
    app.run()
```

```python
from Modules import Modules

class AppName
    __init__(self, flags...):
        # load config file
        # declare global variables
        self.module_one = ModuleOne(vars )
        self.module_two = ModuleTwo(vars )

    run( ):
        # execute all methods needed
        self.module_one.method()
        self.module_two.method()
```

## module.py

```python
class ModuleOne
    __init__(self, flags...):
        #Declare global variables
    method_one():
        #Declare local variables
        # Script


    method_two
        #Declare local variables
        # Script
```