

Une première application



Une première application

Déroulé du cours

- Une première application
- Routes et templates
- Penser les données par un ORM: création des objets
- Penser les données par un ORM: implémentation et interactions
- Interactions avec les utilisateurs: les formulaires
- Gestion des utilisateurs
- TP et bonnes pratiques

Notions Python principales

- Classes : `class`
- Fonctions : `def`
- Décorateurs : `@`
- Algorithmie de base : `if ... then ... else...` , `for ... in...` , `while ...`
- Packages et modules : `from ... import ...`

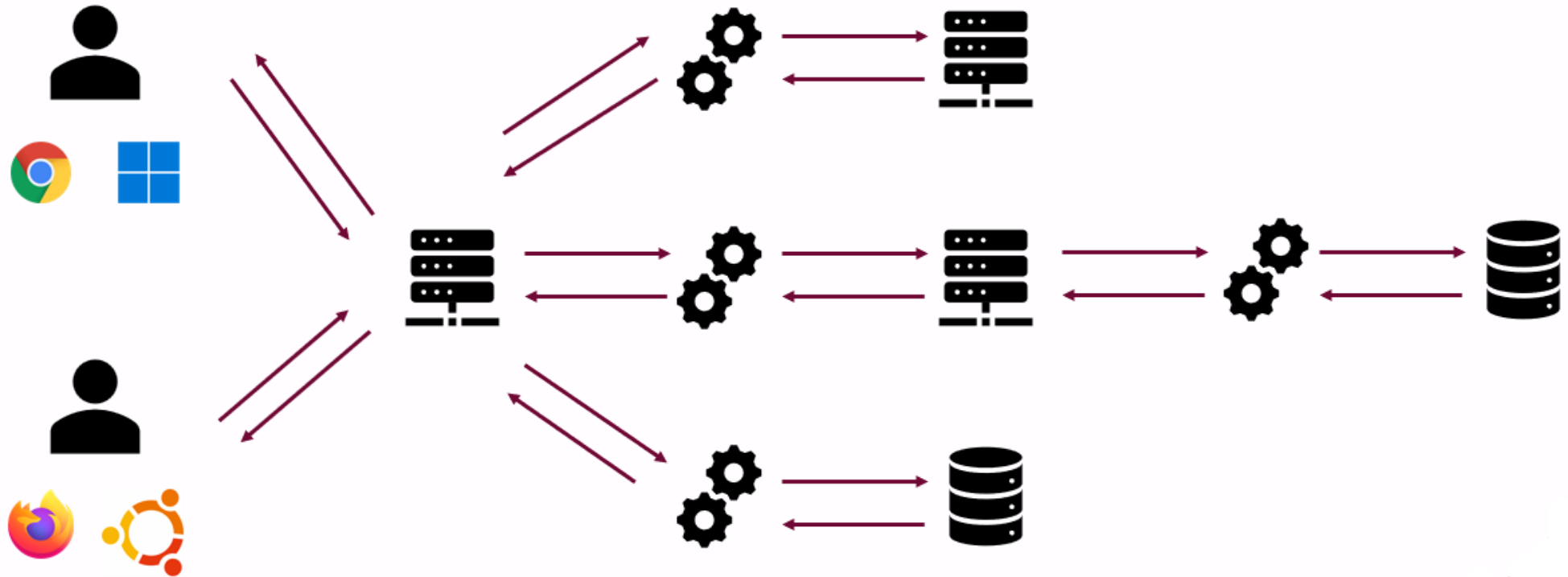
Exercices

- Au moins 1 par personne à l'issue de chaque cours
- Note de participation globale
- Envoi par **mail**
- 3 types d'exercices:
 - Révision **Révision**
 - Approfondissement **Approfondissement**
 - Réflexions sur le cours suivant **Réflexions**

Evaluation

Les bases du Web

Séparation du front-end et du back-end



Une première application

Internet ou Web

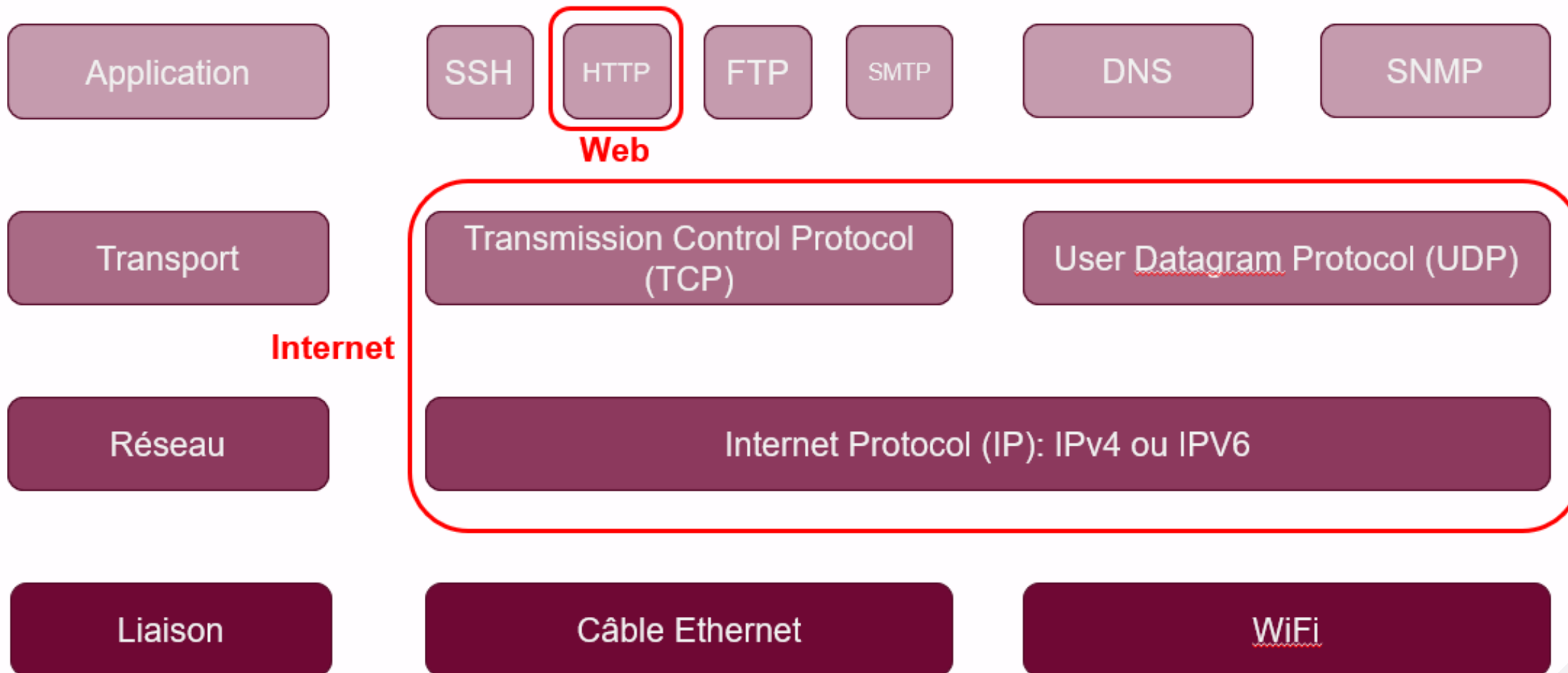
Sir Tim Berners-Lee: inventeur du Web



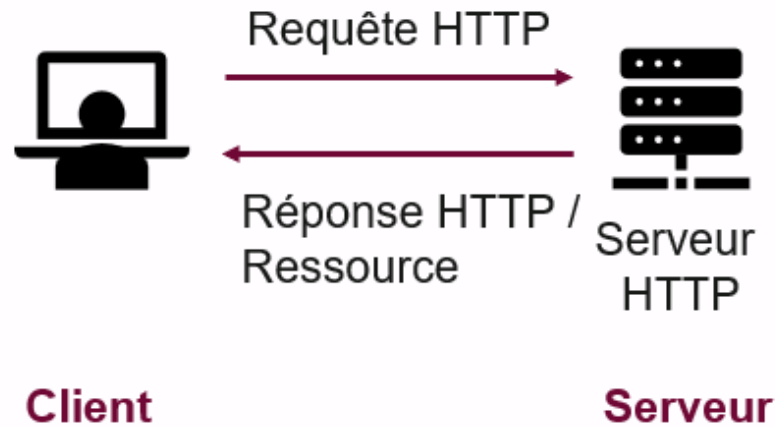
Vinton Cerf:
inventeur de
TCP/IP, pionnier
d'Internet

Une première application

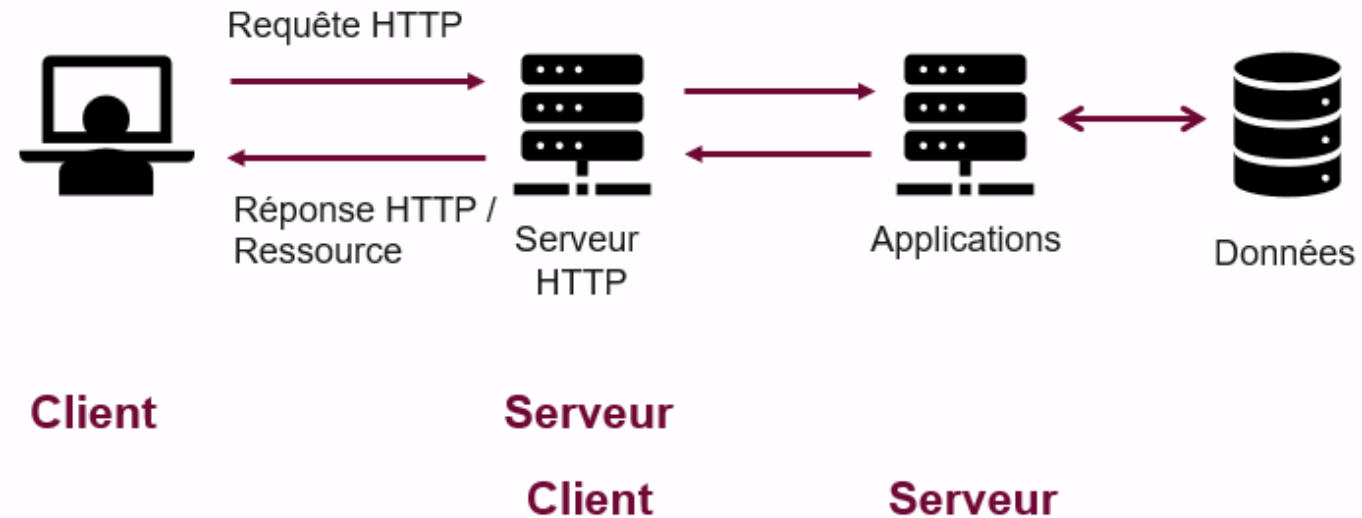
Internet



HTTP



Une architecture à 2 niveaux.
Le serveur écoute les requêtes qui lui arrivent,
et répond en renvoyant une ressource HTTP



Une architecture à 3 niveaux.
Le serveur écoute les requêtes qui lui arrivent, et demande à
des applications de lui renvoyer la/les ressource(s)
nécessaires. Le serveur est alors client des applications.

HTTP: URL

https://www.ina.fr:8080/accueil.html?locale=fr#foot

Protocole	Nom de domaine	Port	Chemin vers la ressource sur le serveur	Paramètres	Ancre
-----------	----------------	------	---	------------	-------

HTTP: méthodes

Méthode	Action
GET	Demande une représentation de la ressource spécifiée
POST	Agi sur la ressource
PUT	Ajoute une ressource
DELETE	Supprime la ressource
PATCH	Modifie une partie de la ressource

HTTP: codes d'erreur

HTTP Status Codes



HTTP: codes d'erreur

Code	Signification
200	<i>OK</i> Tout s'est bien passé
302	<i>Found</i> Document déplacé temporairement
401	<i>Unauthorized</i> Accès non permis
404	<i>Not found</i> La ressource n'existe pas
500	<i>Internal Servor Error</i> Erreur serveur générique
502	<i>Bad Gateway</i> Réponse trop longue à arriver

Premier développement Flask

Flask

Un micro-framework de développement Web Python

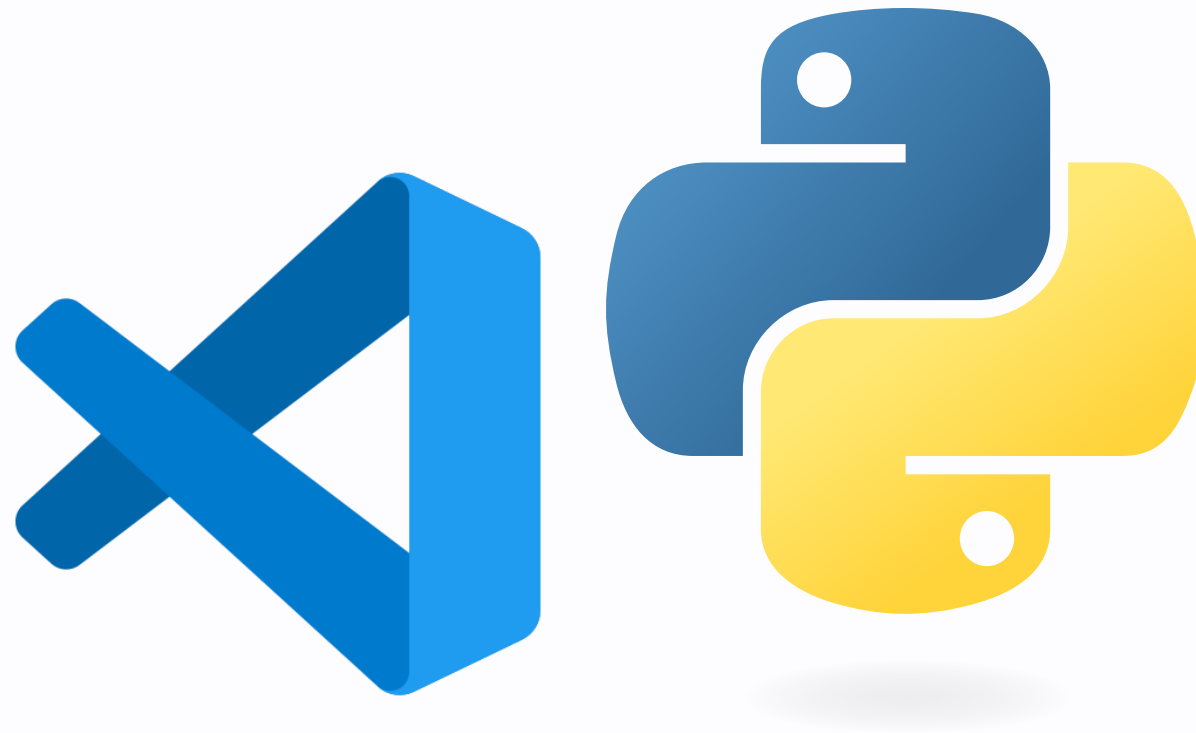


[Lien vers la documentation officielle](#)

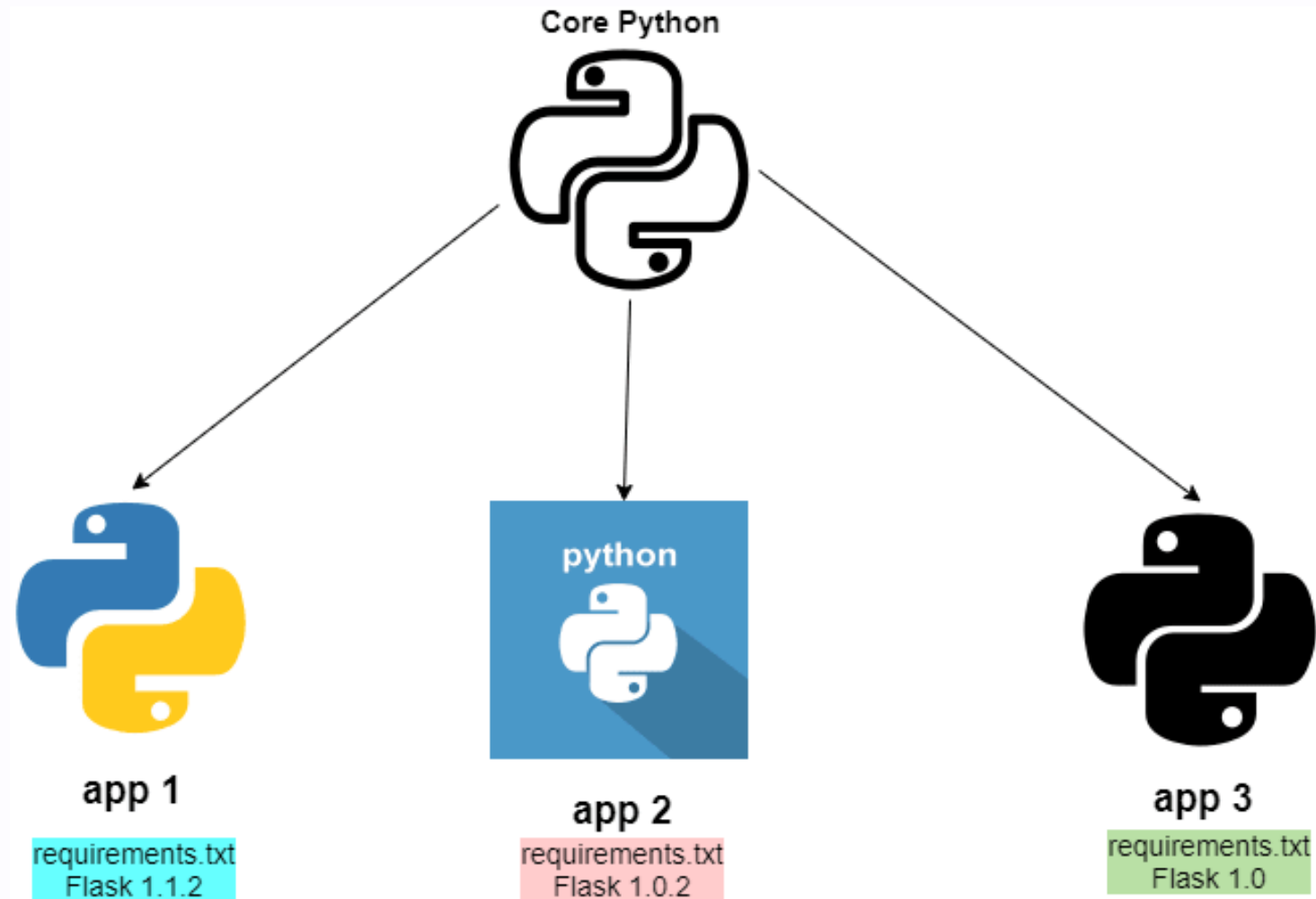
Une structure prédéfinie

```
mon_application/  
├── app/  
│   ├── __init__.py  
│   ├── app.py  
│   ├── models/  
│   ├── routes/  
│   ├── templates/  
│   ├── statics/  
│   └── config.py  
├── tests/  
├── env/  
├── .gitignore  
└── .env
```

Installations



Environnement virtuel



Environnement virtuel

Installation

```
cd dossier_app/  
pip install virtualenv  
virtualenv env -p python3
```

...

```
created virtual environment CPython3.8.6.final.0-64 in 1211ms  
creator CPython3Windows(dest=XXX, clear=False, global=False)  
seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=  
bundle, via=copy, app_data_dir=XXX)  
added seed packages: pip==20.2.2, setuptools==49.6.0, wheel==0.35.1  
activators BashActivator,BatchActivator,FishActivator,  
PowerShellActivator,PythonActivator,XonshActivator
```

Environnement virtuel

Lancement

```
# Ubuntu  
source env/bin/activate  
  
# Windows  
source env/Scripts/activate  
  
# Mac OS  
source env/bin/activate
```

Environnement virtuel

Arrêt

```
# Ubuntu, Windows, Mac OS  
deactivate
```

Installation de Flask

```
# dans l'environnement de travail  
pip install flask
```

```
...
```

```
Installing collected packages: colorama, click, itsdangerous, MarkupSafe,  
    Jinja2, Werkzeug, zipp, importlib-metadata, flask  
Successfully installed Jinja2-3.1.2 MarkupSafe-2.1.1 Werkzeug-2.2.2  
    click-8.1.3 colorama-0.4.5 flask-2.2.2 importlib-metadata-5.0.0  
    itsdangerous-2.1.2 zipp-3.9.0
```

Paquets dans l'environnement

```
pip freeze

...
click==8.1.3
colorama==0.4.5
Flask==2.2.2
importlib-metadata==5.0.0
itsdangerous==2.1.2
Jinja2==3.1.2
MarkupSafe==2.1.1
Werkzeug==2.2.2
zipp==3.9.0
```


Requirements

```
# à la racine du dossier de l'application Flask, exécuter  
# cette commande dès qu'un pip install est exécuté dans  
# l'environnement
```

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

Modularisation

Mise en pratique

Dans un dossier `modules` en dehors de celui de l'application créée précédemment, installer un environnement virtuel

```
| application/  
|   |-- env/  
| modules/  
|   |-- env/
```

Mise en pratique

```
#!/ on se trouve dans application/  
deactivate  
  
cd ../  
  
mkdir modules/ && cd modules/  
  
virtualenv env -p python3
```

Une structure simple de package

```
| nom_package/  
|   |-- __init__.py  
|   code_appelant_package.py
```

Une structure simple de package

```
# __init__.py  
  
une_variable = "var"
```

Une structure simple de package

```
# code_appelant_package.py  
  
from un_package import une_variable  
  
print(une_variable)
```

Un package avec des modules

```
| nom_package/  
| | -- __init__.py  
| | -- module.py  
| code_appelant_package_et_module.py
```


Un package avec des modules

```
## __init__.py  
  
ma_variable = 25
```

Un package avec des modules

```
## module1.py

class Module1():
    def une_fonction(self, un_nombre):
        return un_nombre*2
```

Un package avec des modules

```
# code_appelant_package.py
from package.module1 import Module1
from package import ma_variable

multiplication = Module1().une_fonction(ma_variable)
print(multiplication)
```

Les chemins

```
| package1/  
|   |-- __init__.py  
|   |-- module1.py  
|   |-- module2.py  
| package2/  
|   |-- __init__.py  
|   |-- module1.py  
|   |-- module2.py  
| run.py
```

Première application Flask

Une application en trois lignes

```
from flask import Flask  
  
app = Flask(__name__)  
app.run()
```

Structurer l'application

```
| app/  
|   |-- __init__.py  
|   |-- app.py  
| env/  
| nom_application.py
```

Structurer l'application

```
# app/app.py
from flask import Flask

app = Flask(__name__)
```


Structurer l'application

```
# nom_application.py
from app.app import app

if __name__ == "__main__":
    app.run()
```

Variables d'environnement

```
| app/  
|   |-- __init__.py  
|   |-- app.py  
|   |-- config.py  
| nom_application.py  
| env/  
| .env
```

Variables d'environnement

```
# .env  
DEBUG=True
```

Variables d'environnement

```
# app/config.py
import dotenv
import os

BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
dotenv.load_dotenv(os.path.join(BASE_DIR, '.env'))

class Config():
    DEBUG = os.environ.get("DEBUG")
```

Variables d'environnement

```
# app/app.py
from flask import Flask
from .config import Config

app = Flask(__name__)
app.config.from_object(Config)
```

Variables d'environnement

```
# mon_application.py
from app.app import app

if __name__ == "__main__":
    app.run(debug=app.config["DEBUG"])
```