



Programmation PYTHON

EPSI – Module 3^{ème} année

🎯 OBJECTIFS

- Comprendre la philosophie de Python
- Maîtriser la syntaxe de base du langage
- Mettre en œuvre les modules principaux de python
- Concevoir un projet en python

```
def __init__(self, path):
    self.file = None
    self.fingerprints = set()
    self.logdups = True
    self.debug = debug
    self.logger = logging.getLogger(__name__)
    if path:
        self.file = open(os.path.join(path, 'request_fingerprints.log'), 'a')
        self.file.seek(0)
        self.fingerprints.update(self._get_fingerprints())

    @classmethod
    def from_settings(cls, settings):
        debug = settings.getbool('DEBUG', False)
        return cls(job_dir(settings), debug)

    def request_seen(self, request):
        fp = self.request_fingerprint(request)
        if fp in self.fingerprints:
            return True
        self.fingerprints.add(fp)
        if self.file:
            self.file.write(fp + os.linesep)

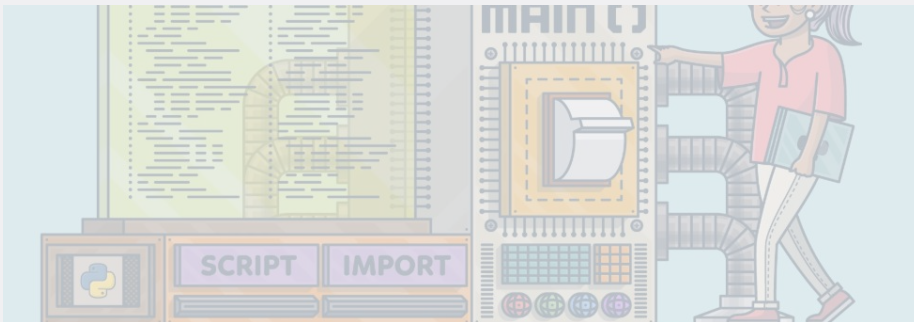
    def request_fingerprint(self, request):
        return request_fingerprint(request)
```



Les bases du langage



Programmation Objet en Python



Utiliser les bibliothèques standards



Structurer, concevoir et refactoriser une application

Pré-requis

- Installer anaconda (python 3.7) :
<https://www.anaconda.com/distribution/>
- Installer l'IDE PyCharm pour Anaconda
 - PyCharm for Anaconda Community Edition<https://www.jetbrains.com/pycharm/promo/anaconda/>

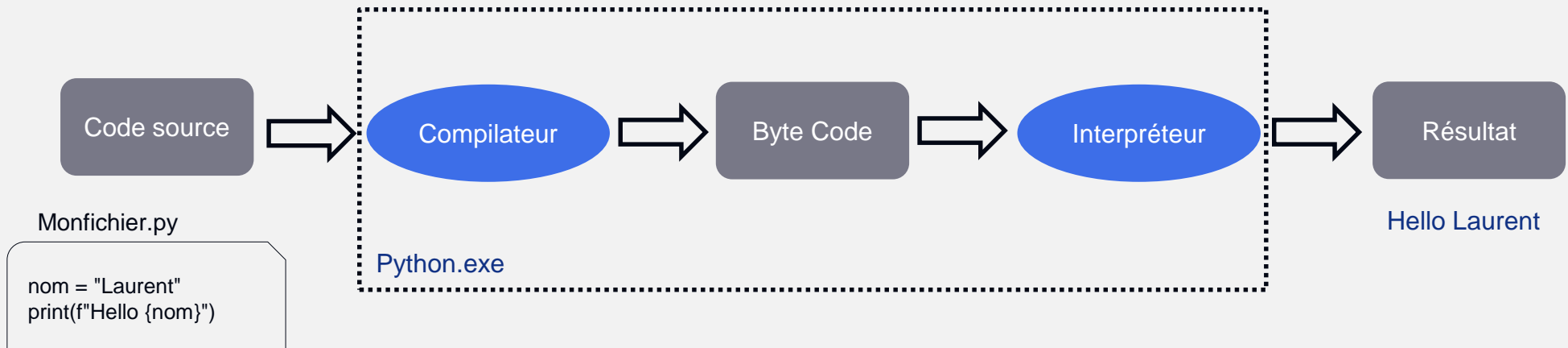
Présentation des caractéristiques du langage

- Conçu en 1989 par Guido van Rossum
- Open source : <https://www.python.org/downloads/source/>
- Multi-plateforme : Linux, Mac, Windows, Raspberry
- Gratuit
- Orienté objet
- Précompilé ET Interprété
- Typage Dynamique
- Bibliothèque standard très complète
- Version actuelle : 3.7.4
- Python 2.7, plus maintenu en 2020

Comment Python est utilisé ?

- Développement Web
 - Django, Flask
- Scripting et automatisation
- Calcul scientifiques
 - Numpy, SciPy, Pandas et Matplotlib
- Machine Learning, Big Data
 - PyTorch, TensorFlow, Scikit-learn
- Application mobile, jeux
 - Kivy
- Embarqué
 - Raspberry, micro python, pyboard

Langage interprété ou compilé



```
Python 3.7.3 (default, Mar 27 2019, 17:13:21) [MSC v.1915 64 bit (AMD64)]
```

```
Python 3.7.3 (default, Mar 27 2019, 17:13:21) [MSC v.1915 64 bit (AMD64)] on win32
```

```
In[2]: def myfunc(alist):  
...:     return len(alist)
```

```
...: import dis
```

```
In[3]: dis.dis(myfunc)
```

```
2           0 LOAD_GLOBAL           0 (len)  
           2 LOAD_FAST              0 (alist)  
           4 CALL_FUNCTION             1  
           6 RETURN_VALUE
```

Les types de base

- Type numérique et booléen

int

0 78 -5

float

5.8 -3.9 1.45e-7

complexe

5 + 7j

bool

True False

- Type itérable

str

"anom" 'tel'

list

[1,5,6] ["a", "b"]

tuple

(1,5,6)

set

{1,5,6}

dict

{"name": "Laurent", "student_id": 123}

Les opérateurs de base

addition, soustraction,
multiplication et division

+, -, *, /, //

Puissance, carré, valeur abs

**, pow, sqrt, abs

comparaisons

==, is, !=, is not, >, >=, <, <=

logique

or, and, not

Affectation, typage, référence objet

// C# ou Java

int age = 46;

String name = "LNF";

Python

age = 46

name = "LNF"

age2 = age

id(age) # 2910253770376

id(age2) # 2910253770376

Conversion de type et Built-in Functions

pi = 3.14159

angle = 45

Pi + angle = 48.14159 # conversion implicite

“Angle : “ + str(angle) + “ dg” == “Angle : 45 dg”

int(pi) == 3

float(angle) == 45.0

`cours_csharp = False`

`int(cours_csharp) == 0`

`cours_python = True`

`int(cours_python) == 1`

`alimente_cours = None`

Traitements des chaines de caractères

`'Hello World' == "Hello World"`

`"hello".capitalize() == "Hello"`

`"hello".replace("e", "a") == "hallo"`

`"hello".isalpha() == True`

`"123".isdigit() == True`

`"some, csv, values".split(",") == ["some", "csv", "values"]`

`nom = "bob"`

`prenom = "sinclair"`

`f"Salut {nom} – {prenom}"`

`>> Salut bob - sinclair`

Extraction de fragments de chaines

resultat = "Intervention"

```
resultat = "intervention"[0:4] # inte
resultat = "intervention"[0:6:2] # itr (par pas de 2)
resultat = "intervention"[-3:] # ion (démarrage à -3 de la fin)
resultat = "intervention"[::-1] # noitnevretni (inversion des lettres en commençant par la fin)
```

```
mon_entree = input("entrer un nombre")  
  
print(f"Le nombre saisi est {mon_entree}")
```

Exercice



- Exercice de base n°1 et n°2
- https://github.com/Inyffels/Code_Epsi/blob/master/Exercices/en_nonce.md



if condition:

print(“condition est vrai”)

print(“Fin”)

if condition:

bloc 1

print(“condition est vrai”)

else:

bloc 2

print(“condition est fausse”)

if-elif-else

```
if choix == 1:  
    # bloc 1  
elif choix == 2:  
    # bloc 2  
elif choix == 3:  
    # bloc 3  
else:  
    # bloc 4
```

if compacté

if $x > 0$:

$y = x^{**}2$

else:

$y = x/4$

$y = x^{**}2$ if $x > 0$ else $y = x/4$

Exercice



- **Exercice 3 : Structures conditionnelles**



while

```
while <test1>:  
    <blocs d'instructions 1>  
    if <test2>: break  
    if <test3>: continue
```

- **break** : sort de la boucle
- **continue** : saute une étape

Exercice



- Exercice 4 : boucle while



Définir une fonction de base en Python

```
def <nom_function>(arg1, arg2...):  
    <bloc d'instructions>  
    return <valeur(s)>
```

```
def ma_fonction(a, b=3, c=1):  
    return (a * b) / c
```

```
ma_fonction(2)    # 6.0
```

Fonction – retour multiple

```
def ma_fonction(a, b, c):  
    result = (a * b) / c  
    e, d = str(result).split('.')  
    return e, d
```

```
e, d = ma_fonction(76, 5, 7)  
print(f"Partie entiere: {e} - Partie décimale: {d[:3]}")
```


Exercice



- **Exercice 5 : fonction et manipulation de chaîne de caractères**



Combinaison de type et « type hinting »

```
Python 3.7.3 (default, Mar 27 2019, 17:13:21) [MSC v.1915 64 bit (AMD64)]
Python 3.7.3 (default, Mar 27 2019, 17:13:21) [MSC v.1915 64 bit (AMD64)] on win32
In[2]: def additionner(a, b):
...:     return a+b
...:
In[3]: resultat = additionner(4, "azerty")
Traceback (most recent call last):
  File "C:\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py", line 3296, in run_code
    exec(code_obj, self.user_global_ns, self.user_ns)
  File "<ipython-input-3-b8a4a3ea6c8c>", line 1, in <module>
    resultat = additionner(4, "azerty")
  File "<ipython-input-2-7b7b0cd2b7cf>", line 2, in additionner
    return a+b
TypeError: unsupported operand type(s) for +: 'int' and 'str'

In[4]: |
```

```
def additionner(a: int, b: int) -> int:
    return a + b
```

Exception

```
try:
    # ... instructions à protéger
except type_exception_1:
    # ... que faire en cas d'erreur de type type_exception_1
except (type_exception_i, type_exception_j):
    # ... que faire en cas d'erreur de type type_exception_i ou type_exception_j
except type_exception_n:
    # ... que faire en cas d'erreur de type type_exception_n
except:
    # ... que faire en cas d'erreur d'un type différent de tous les précédents types
else:
    # ... que faire lorsque une erreur aucune erreur n'est apparue
```

■ lever une exception

```
if value == 10
    raise ValueError
```

Exercice



- Exercice 6 : try / except sur fonction inverse



Liste

Initialisation

```
Ist = []
```

```
IstNumber = [10, 20, 30, 40]
```

```
IstNumberType: List[int] = [3, 6, 8, 34, 10, 567]
```

```
IstHybrid = [34, 'azerty', (4, 8), [1, 2, 3]]
```

```
listInitNumber = range(30)
```

```
IstCinqZeros = [0]*5
```

Accès au premier élément

```
elem = IstNumberType[0]
```

Concaténation

```
IstTotal = IstNumber + IstNumberType
```

Copie

```
IstCopie = IstNumberType
```

Liste

lg = len(IstNumberType)	# taille de la liste
IstNumberType.append(7)	# ajoute un élément à la fin de la liste
IstNumberType.sort()	# trie la liste
IstNumberType.reverse()	# inverse la liste
positionElem = IstNumberType.index(7)	# donne l'indice de l'élément 7 dans la liste
IstNumberType.remove(10)	# retire un élément de la liste
elem = IstNumberType.pop()	# retire le dernier élément de la liste

for

```
for <element> in <objet_sequence>:  
    <blocs d'instructions>  
    if <test1>: break  
    if <test2>: continue
```

```
lst = ['u', 's', 'e', 'r', 1, 5, 3]  
chaine = ""  
for elem in lst:  
    if type(elem) != int:  
        chaine += elem  
    else:  
        break  
print(chaine) # affiche user
```

Exercice



- **Exercice 7 : Liste – Eviter les duplications**



- Sert à renvoyer une liste retravaillée

```
lst = [1, 5, 87, 32, 12]
```

```
new_lst = [x for x in lst]
```

```
new_lst = [x ** 2 for x in lst]
```

```
new_lst = [x ** 2 for x in lst if x < 80]
```

Défis sur la compréhension de liste

■ Défi #1 :

- Soit la liste suivante : `lst = range(30)`
- Afficher en une ligne une nouvelle ligne de nombre pairs sur la base de `lst` en utilisant une liste de compréhension
- `[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29]`
- `=> [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28]`

```
print("Nouvelle liste :", [i for i in lst if i%2 == 0 ])
```

■ Défi #2 :

- Soit la chaîne de caractères suivante : `message = "Python c'est trop puissant"`
- Afficher en une ligne une liste qui met chaque mot en lettres capitales et donne sa longueur
- `=> [('PYTHON', 6), ('C'EST', 5), ('TROP', 4), ('PUISSANT', 8)]`

```
print([(m.upper(), len(m)) for m in message.split(" ")])
```

Focus sur Map

```
listeEntier = [56, 2, 78, 989, 34, 16, 52]
```

```
def cube(x):  
    return x ** 3
```

```
newList = list(map(cube, listeEntier))  
for elem in newList:  
    print(elem)
```

```
# Alternative avec une compréhension de liste  
newList = [cube(x) for x in listeEntier]  
for elem in newList:  
    print(elem)
```

Dictionnaire

```
# initialisation
dicoVide = {}
dicoVide2 = dict()
dicolnit = {"nom": "Bob", "age": 42, "identifiant": "S611456"}
```

```
# Ajout clé/valeur
dicoVide["MaCle"] = "valeur1"
print(dicoVide["MaCle"])
age = dicolnit["age"]
```

```
# copie de dictionnaire
dicoCopie = dicolnit
```

```
# obtention des clés et valeurs
keys = dicoCopie.keys()
values = dicoCopie.values()
if "identifiant" in dicoCopie:
    print(dicoCopie["identifiant"])
```

```
# parcours d'un dictionnaire
for key,value in dicoCopie.items():
    print(f"clé : {key} , valeur : {value}")
```

EXERCICE

Exo 8 : Score au Scrabble

Dans ce jeu à chaque lettre est associé des points. Le score pour un mot est la somme des points des lettres du mot.

Points associés à chaque lettre :

1 point	A, E, I, L, N, O, R, S, T et U
2 points	D et G
3 points	B, C, M et P
4 points	F, H, V, W et Y
5 points	K
8 points	J et X
10 points	Q et Z

Créer un programme qui pour un mot saisi renvoie le score.

Fonction (*args)

```
def additionne(i, *args):  
    somme = i  
    print(args[0])  
    for val in args:  
        somme+=val  
    return somme
```

```
somme = additionne(6,4,5,10)
```

```
# arg[0] = 4  somme = 25
```

Fonction (**kwargs)

```
def fonction(**kwargs):  
    return kwargs['a'] + 2*kwargs["a"]*kwargs["b"] + kwargs["c"]  # a + 2ab + c
```

```
dico = {'a':3, 'b':4, 'c':2}  
print(fonction(**dico))
```

```
def afficher_recette(nom, **kwargs):  
    print(f"=== {nom} ===")  
    for cle in kwargs:  
        print(f"{cle} : {kwargs[cle]}")
```

```
ingredients_tiramisu = {"beurre": "100g", "cafe": "10cl", "oeufs": 3}  
afficher_recette("Tiramisu", **ingredients_tiramisu)
```

Fonction qui retourne un objet de type fonction

```
multiply = lambda a, b : a*b
```

```
result = multiply(2,5)
```

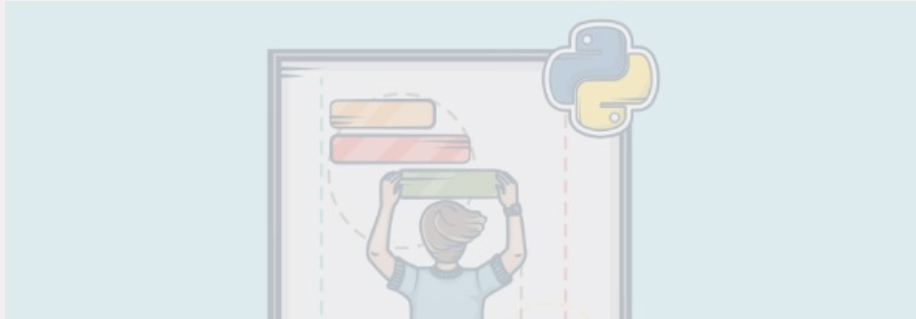
```
print(result)
```

Expression retournée



liste de paramètres

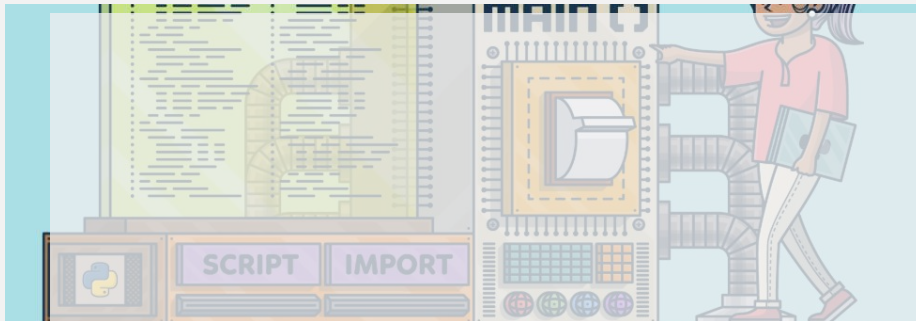




Les bases du langage



Programmation Objet en Python



Utiliser les bibliothèques standards



Structurer, concevoir et refactoriser une application

Classe, Constructeur, Instance

```
class Utilisateur:
    def __init__(self, identifiant, nom, motDePasse):
        self.identifiant = identifiant
        self.nom = nom
        self.motDePasse = motDePasse

    def controler_mot_de_passe(self, password):
        return self.motDePasse == password

greg = Utilisateur("S611234", "Roller", "azer@ty")
greg.controler_mot_de_passe("toto")  # False
```

Visibilité, protéger des attributs et méthode

```
class Utilisateur:
```

```
    def __init__(self, identifiant, nom, motDePasse):
```

```
        self.identifiant = identifiant
```

```
        self.nom = nom
```

```
        self.__motDePasse = motDePasse
```

```
        self.__nom_interne = None
```

```
    def controler_mot_de_passe(self, password):
```

```
        return self.__motDePasse == password
```

```
    def __concatener_elements(self):
```

```
        self.__nom_interne = self.identifiant + "-" + self.nom
```

```
greg = Utilisateur("S611234", "Royer", "azer@ty")
```

```
greg.controler_mot_de_passe("toto") # False
```

```
mot = greg.__motDePasse # AttributeError: 'Utilisateur' object has no attribute '__motDePasse'
```

```
greg.__concatener_elements() # AttributeError: 'Utilisateur' object has no attribute  
'__concatener_elements'
```

Visibilité, Getter et Setter

```
class Vector:
    def __init__(self, x, y):
        self.__x = x
        self.__y = y

    @property
    def x(self):
        return self.__x

    @property
    def y(self):
        return self.__y

    @x.setter
    def x(self, val):
        if val > 0:
            self.__x = val

    @y.setter
    def y(self, val):
        if val > 0:
            self.__y = val
```

```
if __name__ == '__main__':
    v1 = Vector(2,5)
    v2 = Vector(1,4)

    print (v1.x)
    v1.x = 8
```

Méthodes spéciales

```
class Vector:
    def __init__(self, x, y):
        self.__x = x
        self.__y = y

    def __add__(self, other):
        x = self.__x + other.x
        y = self.__y + other.y
        return Vector(x,y)

    def __repr__(self):
        return f'(x={self.__x}, y={self.__y})'
```

```
if __name__ == '__main__':
    v1 = Vector(2,5)
    v2 = Vector(1,4)

    print(v1)
    print(v2)
    v3 = v1 + v2
    print(v3)
```

Méthodes statiques et méthode de classes

```
import uuid

class Personne:

    friends = []

    def __init__(self, nom):
        self.nom = nom

    @classmethod
    def ajouter_ami(cls, ami):
        cls.friends.append(ami) # Accès juste aux propriétés

    @staticmethod
    def generer_uuid():
        # pas d'accès aux méthodes et propriétés de la classe ou instance
        return str(uuid.uuid4())
```

```
pers = Personne("Inf")
Personne.ajouter_ami("bob")
guid1 = Personne.generer_uuid()
```

HERITAGE

```
class Personne:
    def __init__(self, nom, age=18):
        self._nom = nom
        self._age = age

    def _is_adulte(self):
        return self._age >= 18
```

```
class Invite(Personne):
    def __init__(self, nom, age, entreprise):
        super().__init__(nom, age)
        self.entreprise = entreprise

    def __str__(self):
        if self._is_adulte():
            return f"GUEST : {self._nom}, âge: {self._age}, Entreprise : {self.entreprise}"
        else:
            return "invité mineur"
```

HERITAGE : Surcharge de méthode

```
class Personne:
    def __init__(self, nom, age=18):
        self._nom = nom
        self._age = age

    def _is_adulte(self):
        return self._age >= 18
```

```
class Invite(Personne):
    def __init__(self, nom, age, entreprise):
        super().__init__(nom, age)
        self.entreprise = entreprise

    def _is_adulte(self):
        return self._age >= 5

    def __str__(self):
        if self._is_adulte():
            return f"GUEST : {self._nom}, âge: {self._age}, Entreprise : {self.entreprise}"
        else:
            return "invité mineur"
```


Abstraction vs Interface

- Pas d'interface en Python
- Utiliser les méthodes abstraites pour forcer l'implémentation d'une méthode dans une classe fille

```
import abc
```

```
class Vehicule(abc.ABC):  
    @abc.abstractmethod  
    def avancer(self):  
        pass
```

```
class Voiture(Vehicule):  
    def __init__(self):  
        self.reservoir = 40  
  
    def avancer(self):  
        self.reservoir -= 1
```

Exercice



- Exercice : Développer une `EPSI_Exception` qui hérite d'`Exception` et écrit dans un fichier :
 - date / heure
 - exception interne
 - Code erreur spécifique
 - Message erreur spécifique

```
try:  
    a = 5 / 0  
except Exception as exc:  
    raise EpsiException(exc, 10, "attention à la division par zéro")
```

Tips

Ecrire dans un fichier :

```
import os
os.chdir("c:/temp")
mon_fichier = open("monfichier.txt", "a")
mon_fichier.write("ecrire une premiere ligne")
mon_fichier.close()
```

Lire dans un fichier :

```
with open("monfichier.txt", "r") as fichier:
    texte_recupere = fichier.read()
```

TP



- **Gestion de compte bancaire**
 - US et scénario de tests
 - Behaviour Driven Development
 - Bouchenage de service (Mocking)



User Story

c'est quoi une US ?

c'est quoi une user story ?

#ID	Titre	Type : Fonctionnel
Description <i>En tant que...</i> <i>Je veux...</i> <i>Afin de ...</i>		
Règles métier <ul style="list-style-type: none">• Règle 1• Règle 2• Règle 3		
Tests d'acceptance <i>Jeu de données</i> Nom du scénario 1 <i>Etant donné que ...</i> <i>Lorsque ...</i> <i>Alors ...</i> Nom du scénario 2 <i>Etant donné que ...</i> <i>Lorsque ...</i> <i>Alors ...</i> Nom du scénario 3 <i>Etant donné que ...</i> <i>Lorsque ...</i> <i>Alors ...</i>		

BDD

c'est quoi ?

BDD est un langage naturel qui en met en avant des exemples concrets

Permet d'automatiser les tests de scenarios et facilite la mise en place des tests de comportements et de tests unitaires

<https://cucumber.io/docs/gherkin/>

Le langage Gherkin est défini pour décrire les scénarios BDD

Aussi connu dans le monde Agile sous la pratique de Given / When / Then (And).

c'est quoi ?

En complément des US, les scénarios BDD apporte :

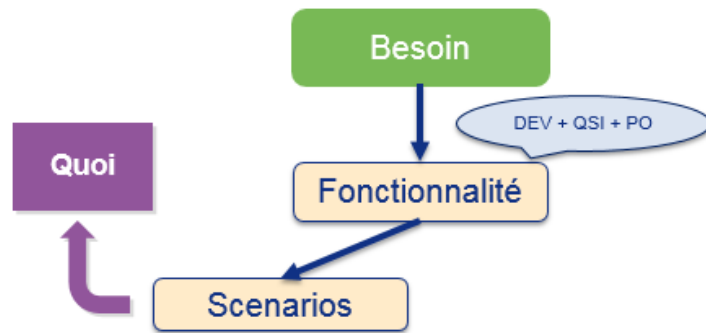
**le contexte (Etant donné que),
l'événement (Lorsque),
et le résultat (Alors)**

Le contexte, l'événement et le résultat sont identifiés pour chaque action de l'utilisateur ou du système.

BDD fonctionne comme une spécification permettant de décrire le comportement du produit

- Settings – Project Interpreter - +
- <https://www.jetbrains.com/help/pycharm/installing-uninstalling-and-upgrading-packages.html>

Écriture des scénarios Gherkin



@RG1

Scenario: Virement simple

Given j'ai un compte cheque avec un solde de 500€

Given j'ai un compte épargne avec un solde de 0€

When j'effectue un virement de 100€ du compte cheque vers le compte épargne

Then le solde du compte cheque est 400€

Then le solde du compte épargne est 100€

Then le virement est confirmé

■ Ecrire en binôme les scénarios illustrant les règles

- RG2
- RG3

Feature: Gestion des Virements

Dans le but de pouvoir gérer mes comptes

En tant que client banque

Je souhaite pouvoir effectuer des virements entre mes comptes

RG1 : virement simple, je vire X€ d'un compte A vers le compte B, le solde est impacté dans les deux comptes.

RG2 : virement hors provision, solde A insuffisant

RG3 : virement plafonné



Correction : Activité pédagogique

Feature: Gestion des Virements

Dans le but de pouvoir gérer mes comptes

En tant que client banque

Je souhaite pouvoir effectuer des virements entre mes comptes

RG1 : virement simple, je vire X€ d'un compte A vers le compte B, le solde est impacté dans les deux comptes.

RG2 : virement hors provision, solde A insuffisant

RG3 : virement plafonné

@RG2

Scenario: Virement hors provision

Given j'ai un compte cheque avec un solde de 50€

Given j'ai un compte épargne avec un solde de 1000€

When j'effectue un virement de 100€ du compte cheque vers le compte épargne

Then le solde du compte cheque est 50€

Then le solde du compte épargne est 1000€

Then le virement est refusé pour motif hors provision

@RG3

Scenario: Virement plafonné

Given j'ai un compte cheque avec un solde de 1000€

Given j'ai un compte épargne avec un solde de 0€

Given la limite de virement est 500€

When j'effectue un virement de 501€ du compte cheque vers le compte épargne

Then le solde du compte cheque est 1000€

Then le solde du compte épargne est 0€

Then le virement est refusé pour motif plafond dépassé

Scenarios GHERKIN

Feature: Crediter_compte

""

EN TANT QUE client de la banque
JE PEUX créditer mon compte courant
AFIN DE l'alimenter
""

Scenario: credit_simple

Enter steps here

Given mon compte courant_123 a un solde de 0 euros
When je credite mon compte courant_123 de 150 euros
Then le solde de mon compte courant_123 est de 150 euros

Feature: virer_compte_simple

""

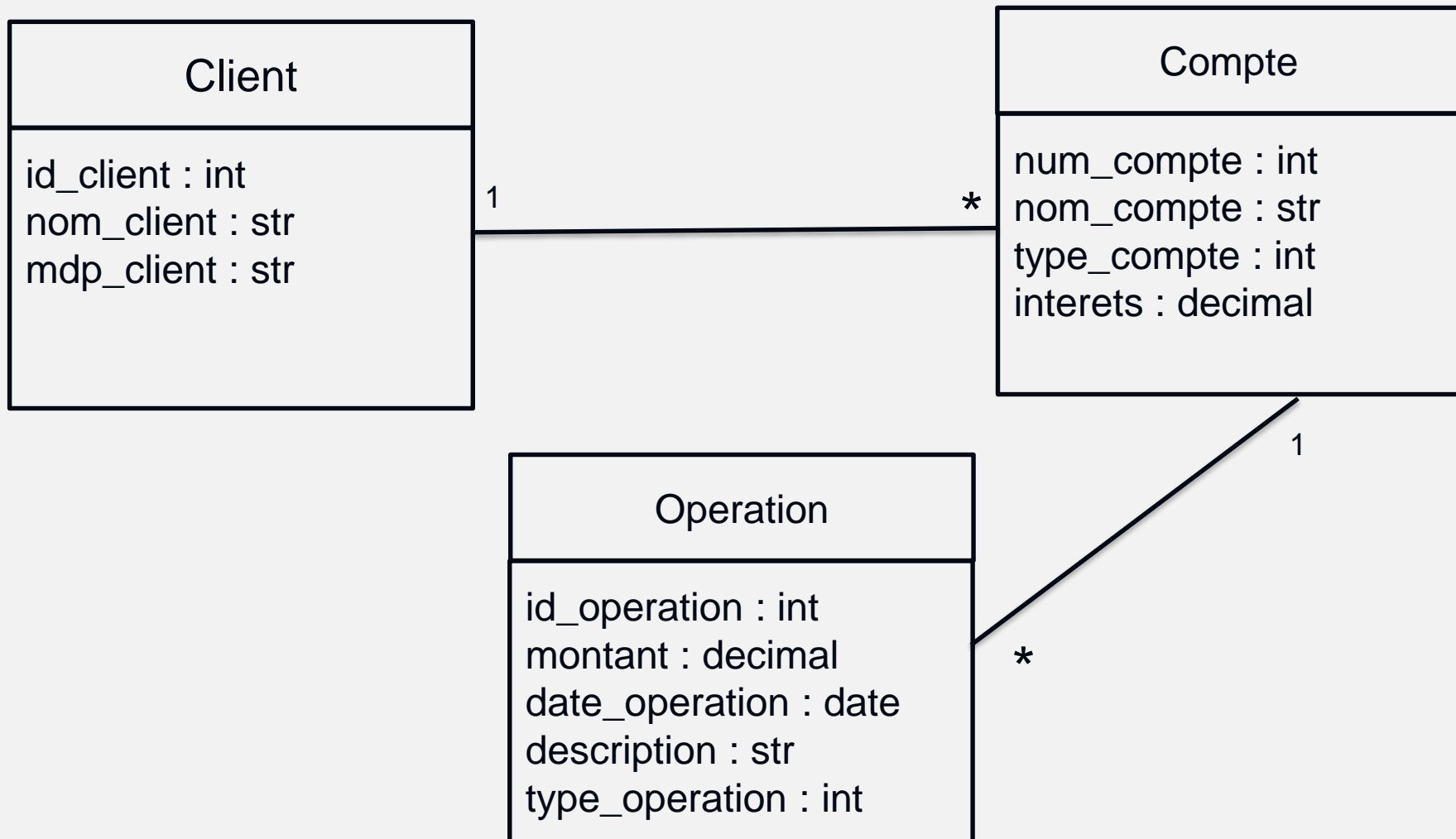
EN TANT QUE client de la banque
JE PEUX effectuer un virement depuis mon compte courant
AFIN DE d'alimenter un compte épargne
""

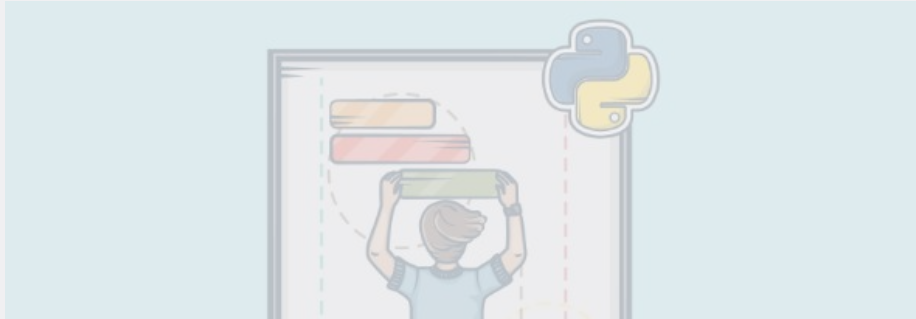
Scenario: virement_simple

Enter steps here

Given mon compte courant_123 a un solde de 280.50 euros
Given mon compte epargne_CEL a un solde de 1200.00 euros
When je vire 100 euros de mon compte courant_123 vers mon compte epargne_CEL
Then alors mon compte courant_123 a un solde de 180.50
And mon compte epargne_CEL a un solde de 1300.00 euros

Modèle à implémenter

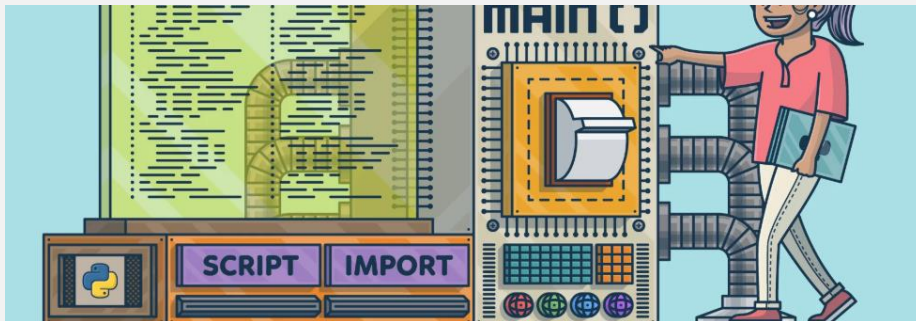




Les bases du langage



Programmation Objet en Python



Utiliser les bibliothèques standards



Structurer, concevoir et refactoriser une application

JSON

Données en json

```
jsonData = '{"nom":"Ln", "age":45, "ville":"Lille"}'
```

charge un dictionnaire python

```
dico = json.loads(jsonData)
```

```
print(dico["ville"]) # affiche Lille
```

```
dicoAdresse = {
```

```
    "rue" : "la joie",
```

```
    "cp" : 59800,
```

```
    "ville" : "Lille",
```

```
    "tels" : ["078494949", "03204848484"]
```

```
}
```

Récupère un json (format str) depuis un dictionnaire

```
jsonAdresse = json.dumps(dicoAdresse)
```

SQL Lite 3

- **BDD Relationnelle utilisée pour l'embarqué (Sur un terminal)**
- **PAS une base de données serveur (non multi-threadé)**
- **Instructions SQL complète**
- **Légère : 600 ko**
- **Cross-plateforme : Windows, Android, IOS, Linux**
- **Transactionnel**
- **Support de SQLAlchemy**

SQL Lite 3

- **Initialiser une base**

```
import sqlite3  
connexion = sqlite3.connect("maBase.db")  
curseur = connexion.cursor() # Récupération d'un curseur
```

- **Exécuter une requête (CREATE, INSERT, SELECT...)**

```
curseur.execute("""INSERT INTO users(name, age)  
VALUES(:name, :age)""", pers)
```

- **Valider les modifications**

```
connexion.commit()
```


SQL Lite 3

■ Annuler les modifications

```
connexion.rollback()
```

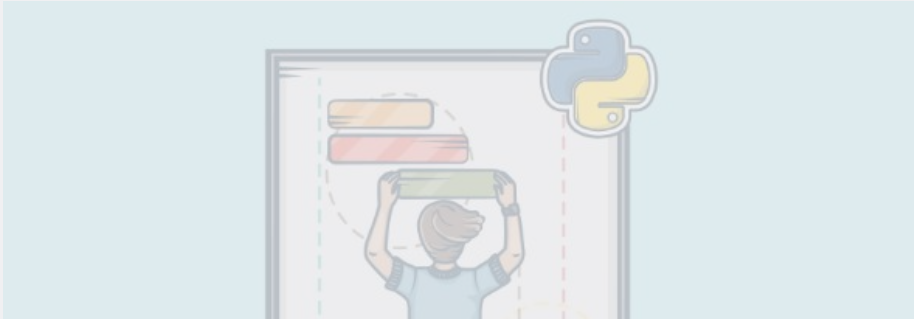
■ Parcourir des enregistrements

```
curseur.execute("SELECT valeur FROM maTable WHERE data = ?", donnee)  
print(curseur.fetchone()) # Retourne un tuple
```

```
curseur.execute("SELECT * FROM clients")  
for row in curseur: # itérer sur le curseur  
    print(row)
```

■ Fermer une connexion

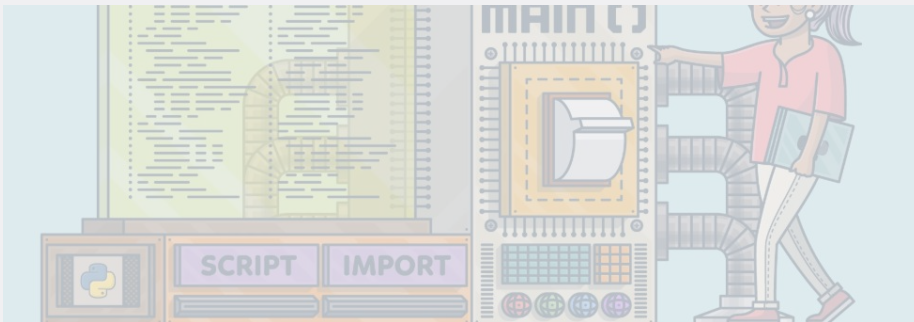
```
connexion.close()
```



Les bases du langage



Programmation Objet en Python



Utiliser les bibliothèques standards



Structurer, concevoir, refactoriser une application

VS



- **Le TDD : Test Driven Development**
- **La revue de code**
- **Le refactoring**

- **Comment testez-vous si la fonction que vous avez écrite «fonctionne» ?**
- **Qu'entendez-vous par « fonctionner » ?**
- **TDD vous oblige à énoncer clairement votre objectif avant d'écrire votre code.**
- **Le « mantra » du TDD est "Test first, code later"**

Exercice Tests unitaires

```
import unittest
from demo3 import Vector

class TestVector(unittest.TestCase):

    def test_add_two_vector(self):
        v1 = Vector(2,5)
        v2 = Vector(1,4)
        v3 = v1 + v2
        self.assertEqual(str(v3), str(Vector(3,9)))

if __name__ == '__main__':
    unittest.main()
```

1. Passe tous ses tests

Vous savez que votre code est « sale » lorsque seulement 95% de vos tests sont passés. Vous savez que vous êtes « mal engagé » lorsque vous testez la couverture est de 0%.

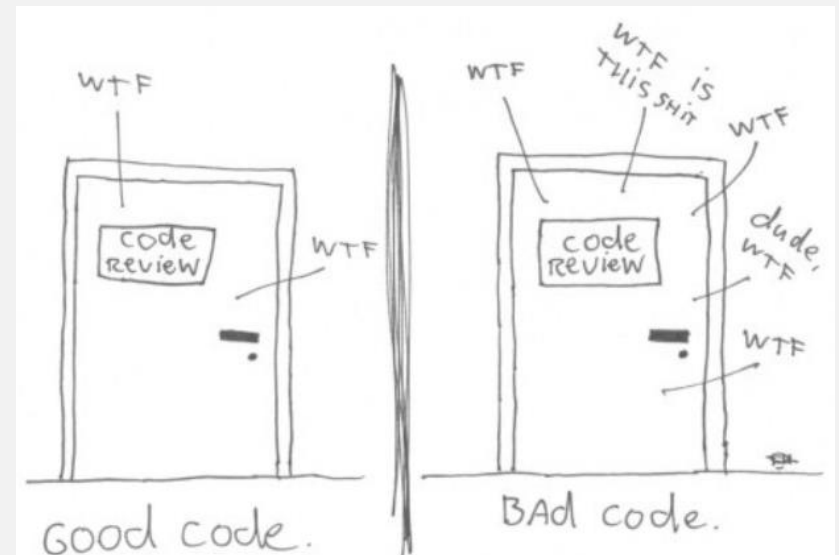
3. Minimum en nombres de classes, en longueur de méthodes...

Moins de code = Moins de bugs. Moins de code = Moins de maintenance à réaliser. Keep it short and simple.

2. Ne contient aucune duplication

Plus il y a de code dupliqué, plus il y a de code à maintenir et plus les évolutions sont risquées.

4. Exprime l'intention





- ➔ **La qualité structurelle du code influe sur la qualité fonctionnelle du produit.**
- ➔ **Les développeurs vivent « dans » le code. Les développeurs passent plus de temps à lire du code qu'à en produire**
- ➔ **Maintenir la maintenabilité**
- ➔ **Contrôler la dette technique**

Quel Code Smell connaissez-vous ?

Code Smell	Technique de refactorisation
Long Method	Extract Method
	Replace Temp with Query
	Introduce Parameter Object
	Preserve Whole Object
	Replace Method with Method Object
	Decompose Conditional
Large Class	Extract Class / Subclass
	Extract Interface
	Duplicate Observed Data
Duplicated code	Extract Method
	Pull Up Method
	Form Template Method
Mistake Errors	Replace Error Code with Exception
	Replace Exception with Test

Code Smell	Technique de refactorisation
Naming smell	Remove Double Negation
Long Parameter List	Replace Parameter with Method Call
	Preserve Whole Object
	Introduce Parameter Object
Data Clumps	Extract Class
	Introduce Parameter Object
	Preserve Whole Object
Couplers	Feature Envy
	Message Chains
	Middle Man

Raison du problème

- Mentalement, il est souvent plus facile d'utiliser une méthode existante que de créer une nouvelle méthode.
- *"Mais il n'y a que deux lignes, il n'y a pas d'utilité pour créer une méthode entière juste pour ça ..."*
- Ce qui signifie qu'une autre ligne est ajoutée, et encore une autre, donnant naissance à une méthode pachydermique.

Techniques disponibles

- Extract Method
- Replace Temp with Query
- Preserve Whole Object Query
- Introduce Parameter Object
- Decompose Conditional

Code Smell

```
# make sure the code only runs on mac os x
mrjVersionExists = GetProperty("mrj.version") is not None
osNameExists = GetProperty("os.name").startswith("Mac OS")
if not mrjVersionExists and not osNameExists :
    print("Not running on a Mac OS X system.")
    exit(1)

# do all the logfile setup stuff
currentLoggingLevel = DEFAULT_LOG_LEVEL
errorFile = File(ERROR_LOG_FILENAME)
warningFile = File(WARNING_LOG_FILENAME)
debugFile = File(DEBUG_LOG_FILENAME)

# order of checks is important; want to go with more granular if multiple files exist
if os.path.errorFile.exists():
    currentLoggingLevel = DDLoggerInterface.LOG_ERROR
if os.path.warningFile.exists():
    currentLoggingLevel = DDLoggerInterface.LOG_WARNING
if os.path.debugFile.exists():
    currentLoggingLevel = DDLoggerInterface.LOG_DEBUG

logger = DDSimpleLogger(CANON_DEBUG_FILENAME, currentLoggingLevel, True, True)

# do all the preferences stuff, and get the default color
preferences = Preferences.userNodeForPackage(getClassG())

r = preferences.getInt(CURTAIN_R, 0)
g = preferences.getInt(CURTAIN_G, 0)
b = preferences.getInt(CURTAIN_B, 0)
a = preferences.getInt(CURTAIN_A, 255)
currentColor = Color(r,g,b,a)
```

Solution possible

```
dielfNotRunningOnMacOsX()
connectToLogfile()
connectToPreferences()
getDefaultColor()

# (1)
def dielfNotRunningOnMacOsX():
    mrjVersionExists = GetProperty("mrj.version") is not None
    osNameExists = GetProperty("os.name").startswith("Mac OS")
    if not mrjVersionExists and not osNameExists:
        print("Not running on a Mac OS X system.")
        exit(1)

# (2)
def connectToLogfile():
    currentLoggingLevel = DEFAULT_LOG_LEVEL
    errorFile = File(ERROR_LOG_FILENAME)
    warningFile = File(WARNING_LOG_FILENAME)
    debugFile = File(DEBUG_LOG_FILENAME)
    # order of checks is important; want to go with more granular if multiple files exist
    if errorFile.exists():
        currentLoggingLevel = DDLoggerInterface.LOG_ERROR
    if warningFile.exists():
        currentLoggingLevel = DDLoggerInterface.LOG_WARNING
    if debugFile.exists():
        currentLoggingLevel = DDLoggerInterface.LOG_DEBUG
    logger = DDSimpleLogger(CANON_DEBUG_FILENAME, currentLoggingLevel, True, True)

# (3)
def connectToPreferences():
    preferences = Preferences.userNodeForPackage(GetClass())

# (4)
def getDefaultColor():
    r = preferences.getInt(CURTAIN_R, 0)
    g = preferences.getInt(CURTAIN_G, 0)
    b = preferences.getInt(CURTAIN_B, 0)
    a = preferences.getInt(CURTAIN_A, 255)
    currentColor = Color(r,g,b,a)
```

➔ Code smell : **long method**

```
def getPrice(quantity, itemPrice):  
    basePrice = quantity * itemPrice  
    discountFactor = None  
    if basePrice > 1000:  
        discountFactor = 0.95  
    else:  
        discountFactor = 0.98  
    return basePrice * discountFactor
```

➔ Solution possible

```
def getPrice(quantity, itemPrice):  
    price = basePrice(quantity, itemPrice)  
    return price * discountFactor(price)  
  
def basePrice(quantity, itemPrice):  
    return quantity * itemPrice  
  
def discountFactor(basePrice):  
    if basePrice > 1000:  
        return 0.95  
    else:  
        return 0.98
```

➔ **Problème :**

- ➔ Le résultat d'une expression est stocké dans une variable locale.

➔ **Solution :**

- ➔ Extraire dans une méthode l'expression et retourner le résultat.

➔ **Avantage:**

- ➔ Meilleure lisibilité du code.
- ➔ Permet d'éviter la duplication de code si la ligne qui est remplacée est utilisée dans plusieurs méthodes.

Introduce Parameter Object

Code Smell

```
def getFlowBetween(start, end):  
    result = 0  
    for entry in Entries:  
        if start <= entry.chargeDate  
        and end >= entry.chargeDate:  
            result += entry.value  
    return result
```

Solution possible

```
def getFlowBetween(rangeDate):  
    result = 0  
    for entry in Entries:  
        if rangeDate.include(entry.chargeDate):  
            result += entry.value  
    return result  
  
class DateRange:  
    def __init__(self, startDate, endDate):  
        self.start = startDate  
        self.end = endDate  
  
    def include(self, givenDate):  
        return self.start <= givenDate and self.end >= givenDate
```


➔ Problème :

- ➔ Votre méthode contient un groupe de paramètres qui se répètent.
- ➔ Votre méthode contient trop de paramètres.

➔ Solution :

- ➔ Remplacer ces paramètres par un objet qui fait sens.

➔ Avantage:

- ➔ En remplaçant un groupe de paramètres qui se répètent dans plusieurs méthodes par un objet, on réduit la duplication de code en les rassemblant et en regroupant le comportement lié à ces paramètres au sein de l'objet également.
- ➔ Le code est plus lisible car l'objet est nommé de manière à faciliter la compréhension de la méthode.

Code Smell

```
def calculateFuelCost(journey, car):  
    journeyTime = journey.getTime()  
    averageSpeed = journey.getAverageSpeed()  
    distance = calculateDistanceTravelled(journeyTime, averageSpeed)  
    return car.costPerKilometre() * distance  
  
def calculateDistanceTravelled(journeyTime, averageSpeed):  
    return journeyTime * averageSpeed
```

Solution possible

```
def calculateFuelCost(journey, car):  
    distance = calculateDistanceTravelled(journey)  
    return car.costPerKilometre() * distance  
  
def calculateDistanceTravelled(journey):  
    return journey.getTime() * journey.getAverageSpeed()
```

➔ Explication:

- ➔ Vous obtenez plusieurs valeurs d'un objet, puis les transmettez en tant que paramètres à une méthode.
- ➔ Au lieu de cela, essayez de passer l'objet entier.

➔ Raison du problème:

- ➔ Le problème est que chaque fois avant que votre méthode soit appelée, les méthodes du futur objet de paramètre doivent être appelées. Si ces méthodes ou la quantité de données obtenues pour la méthode sont modifiées, vous devrez trouver soigneusement une douzaine de ces lieux dans le programme et les implémenter dans chacun d'eux.
- ➔ Après avoir appliqué cette technique de refactoring, le code pour obtenir toutes les données nécessaires sera stocké au même endroit - la méthode elle-même.

➔ Avantage:

- ➔ Au lieu d'un mélange de paramètres, vous voyez un seul objet avec un nom compréhensible.
- ➔ Si la méthode a besoin de plus de données d'un objet, vous n'aurez pas besoin de réécrire tous les endroits où la méthode est utilisée - simplement dans la méthode elle-même.

Code Smell

```
if date.before(SUMMER_START) or  
   date.after(SUMMER_END):  
    charge = quantity * winterRate + winterServiceCharge  
else:  
    charge = quantity * summerRate
```

Solution possible

```
if isWinter(date):  
    charge = winterCharge(quantity)  
else:  
    charge = summerCharge(quantity)
```

➔ Explication:

- ➔ Votre code est composé d'une condition peu explicite.
- ➔ L'expression conditionnelle est composée de nombreuses conditions, ainsi que de nombreuses lignes dans la conséquence ou l'alternative (le "then" et le "else").

➔ Raison du problème:

- ➔ Plus un code est long plus il sera long à comprendre. Et ça devient encore plus compliqué quand le code est rempli de conditions.
- ➔ Le temps de comprendre le code résultant de la condition, on en oublie cette condition. Et le temps de comprendre le « else » on en oublie le « then ».

➔ Avantage:

- ➔ En nous forçant à extraire les conditions, la lisibilité du code s'améliore.
- ➔ De cette façon le code est plus expressif. La lecture du code sera plus efficace.

Code Smell

```
class Price:
    def __init__(self, flight):
        self.flight = flight

    def getPrice(self):
        regularPrice = 100
        summerPrice = 120
        winterPrice = 80
        currentMonth = DateTimeProvider.Now().Month

        if currentMonth >= 7 and currentMonth <= 9:
            return self.flight.distance * summerPrice
        elif currentMonth >= 11 or currentMonth <= 2:
            return self.flight.distance * winterPrice

        return self.flight.distance * regularPrice
```

Solution possible

```
class Price:
    def __init__(self, flight):
        self.flight = flight

    def getPrice(self):
        return PriceCalculator(self.flight).compute()

class PriceCalculator:
    def __init__(self, flight):
        self.distance = flight.distance
        self.currentMonth = DateTimeProvider.Now().Month
        self.regularPrice = 100 # Magic number à externaliser
        self.summerPrice = 120
        self.winterPrice = 80

    def isWinterMonth(self):
        return self.currentMonth >= 11 or self.currentMonth <= 2

    def isSummerMonth(self):
        return self.currentMonth >= 7 and self.currentMonth <= 9

    def compute(self):
        if self.isSummerMonth():
            return self.distance * self.summerPrice
        elif self.isWinterMonth():
            return self.distance * self.winterPrice
        return self.distance * self.regularPrice
```

➔ Explication:

- ➔ Votre code est composé d'une longue méthode qui utilise des variables locales de telle manière que vous ne pouvez pas appliquer l'Extract Method.

➔ Raison du problème:

- ➔ Une méthode est trop longue et vous ne pouvez pas la séparer en raison des masses enchevêtrées de variables locales difficiles à isoler les unes des autres.
- ➔ L'idée consiste à isoler la méthode entière dans une classe distincte et à transformer ses variables locales en champs de la classe.

➔ Avantage:

- ➔ Isoler une méthode longue dans sa propre classe permet d'empêcher une méthode de gonfler.
- ➔ Cela permet également de le diviser en sous-méthodes dans la classe destination, sans polluer la classe d'origine avec des méthodes utilitaires.

Raison du problème

Techniques disponibles

- Extract Class / Subclass / interface
- Duplicate Observed Data

➔ Code smell : large class

```
class AdCreative:
    def __init__(self):
        self.isImage = None
        self.adText = None
        self.height = None
        self.width = None
        self.dateCreated = None

    def render(self):
        if self.isImage is True:
            return str.format("<img height='{0}' width='{1}' alt='{2}'/>",
                              self.height, self.width, self.adText)
        return self.adText
```

➔ Solution possible

```
class AdCreative:
    def __init__(self, *args):
        self.isImage = args[0]
        self.adText = args[1]
        self.height = None
        self.width = None

    def render(self):
        return self.adText

class ImageCreative(AdCreative):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

    def render(self):
        if self.isImage is True:
            return "<img height={} width={} alt={}/>".format(self.height,
                                                              self.width, self.adText)
        return self.adText
```

➔ **Problème :**

- ➔ Une classe contient des fonctionnalités qui ne sont utilisées que dans certains cas.

➔ **Solution :**

- ➔ Déplacer dans une sous-classe les méthodes et les champs implémentant un cas d'utilisation rare.

➔ **Avantage:**

- ➔ Respect des principes SOLID.

➔ Explication:

- ➔ Dans une classe, vous constatez un mélange de code métier et de code d'affichage (GUI)

➔ Raison du problème:

- ➔ Mauvaise compréhension de la responsabilité d'une classe. La solution consiste à extraire les informations d'affichage dans une ou plusieurs autres classes. La mise en place du pattern « Observer » peut être une solution appropriée.

➔ Avantage:

- ➔ Séparation des responsabilités entre les classes qui traitent de la logique métier et celles qui portent l'affichage des données (Single Responsibility Principle)
- ➔ Si une nouvelle vue est demandée par le métier, la classe qui porte la logique métier n'est pas impactée. Il suffit que créer une nouvelle vue (Open / Closed Principle)

Raison du problème

- Un mauvais nommage apporte son lot d'incompréhension.
- Evidemment on retrouve les classiques nommage trop court, codifié, ou sans rapport avec le métier.
- Il y a aussi les méthodes avec des noms explicite, mais dont le contexte peut complexifier la compréhension.

Techniques disponibles

- Remove Double Negation

Code Smell

```
def execute(self):
    customer = Customer()
    if not customer.isNotPro():
        print("User isn't not pro ? {}".format(not customer.isNotPro()))

class Customer:
    def __init__(self):
        self.siren = None

    def isNotPro(self):
        return self.siren is None
```

Solution possible

```
def execute(self):
    customer = Customer()

    if customer.isPro():
        print("User is pro : {}".format(
            customer.isPro()))

class Customer:
    def __init__(self):
        self.siren = None

    def isPro(self):
        return self.siren is not None
```

➔ Explication:

- ➔ Il arrive dans le code que l'on ajoute une condition négative dans le nom d'une méthode.
- ➔ La double négation arrive en général à la suite d'une spécification qui indique explicitement la négation.

➔ Raison du problème:

- ➔ La double négation dans une condition entraîne une confusion pour la compréhension.

➔ Avantage:

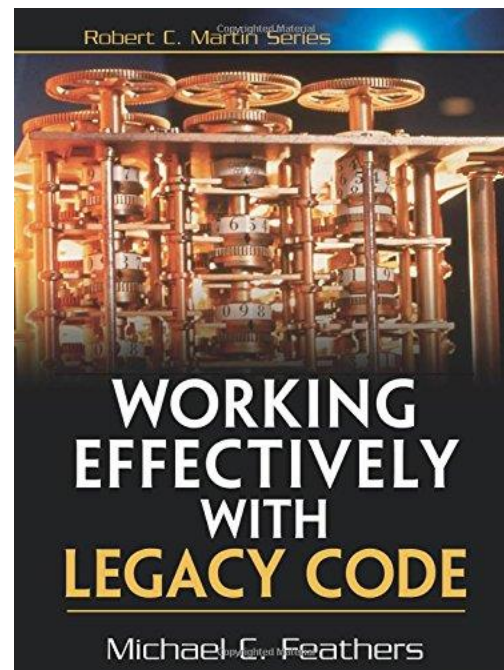
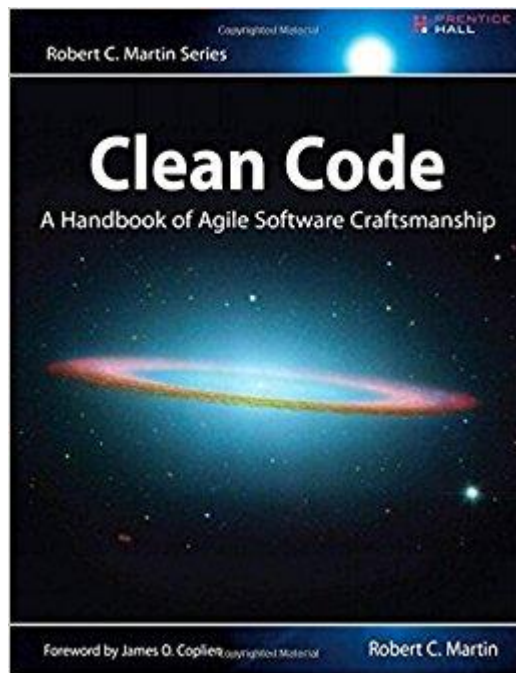
- ➔ Facilite la compréhension en explicitant le code. Cela nous évite des nœuds au cerveau.

Conseil de vétéran : Il vaut mieux 2 méthodes explicites sur des conditions proches plutôt qu'une méthode peu explicite.

Les Tips à retenir

- ➔ **Faites confiance aux outils de votre IDE (Resharper par exemple), solution d'amélioration automatique, complexité cyclomatique**
- ➔ **Les raccourcis claviers**
- ➔ **Ne vous lancer pas dans une refacto en fin de release**
- ➔ **Ne vous lancer pas dans une refacto sans être couvert par des tests unitaires**
- ➔ **Un renomme de variables / méthodes peut parfois changer toutes l'interprétation du code.**
- ➔ **Anglais / Français ==> Créer avec votre métier un glossaire partagé**
- ➔ **Documentez-vous, Pratiquez !**

- [Clean Code : A Handbook of Agile Software Craftsmanship](#)
- [Working effectively with legacy code](#)
- <https://refactoring.guru/>
- <https://app.pluralsight.com/library/courses/writing-clean-code-humans>
- <https://app.pluralsight.com/library/courses/refactoring-fundamentals>



Dirty App



- https://github.com/Inyffels/CodeEpsi/tree/master/dirty_app



Mise en page :

- Une ligne doit contenir 80 caractères maximum.
- L'indentation doit être de 4 espaces.
- Ajoutez deux lignes vides entre deux éléments de haut niveau, des classes par exemple, pour des questions d'ergonomie.
- Séparez chaque fonction par une ligne vide.
- Les noms (variable, fonction, classe, ...) ne doivent pas contenir d'accent. Que des lettres ou des chiffres.
- ...

from Model import Calc

Package



Classe

- **Package** : nom d'un dossier + fichier `__init__.py`
- **Importation absolue versus relative**
- **Absolue** : `from Exemples.TDD.Model.calc import Calc`
- **Relative** : `from ..Domain.Assurance import Contract`

Remonter 2 fois dans l'arborescence depuis la position courante



TP

https://github.com/Inyffels/Code_Epsi/blob/master/Support/EPsi/

■ Projet : Easy SAV

- Objectif : Equiper une flotte de techniciens en service après vente d'une application sur tablette leur permettant de gérer leur interventions à distance.
- Processus métier :

1



Opératrice

- Recevoir des appels clients
- Saisir les interventions

2



Responsable

- Affecter les interventions aux techniciens
- Suivre l'avancer des interventions

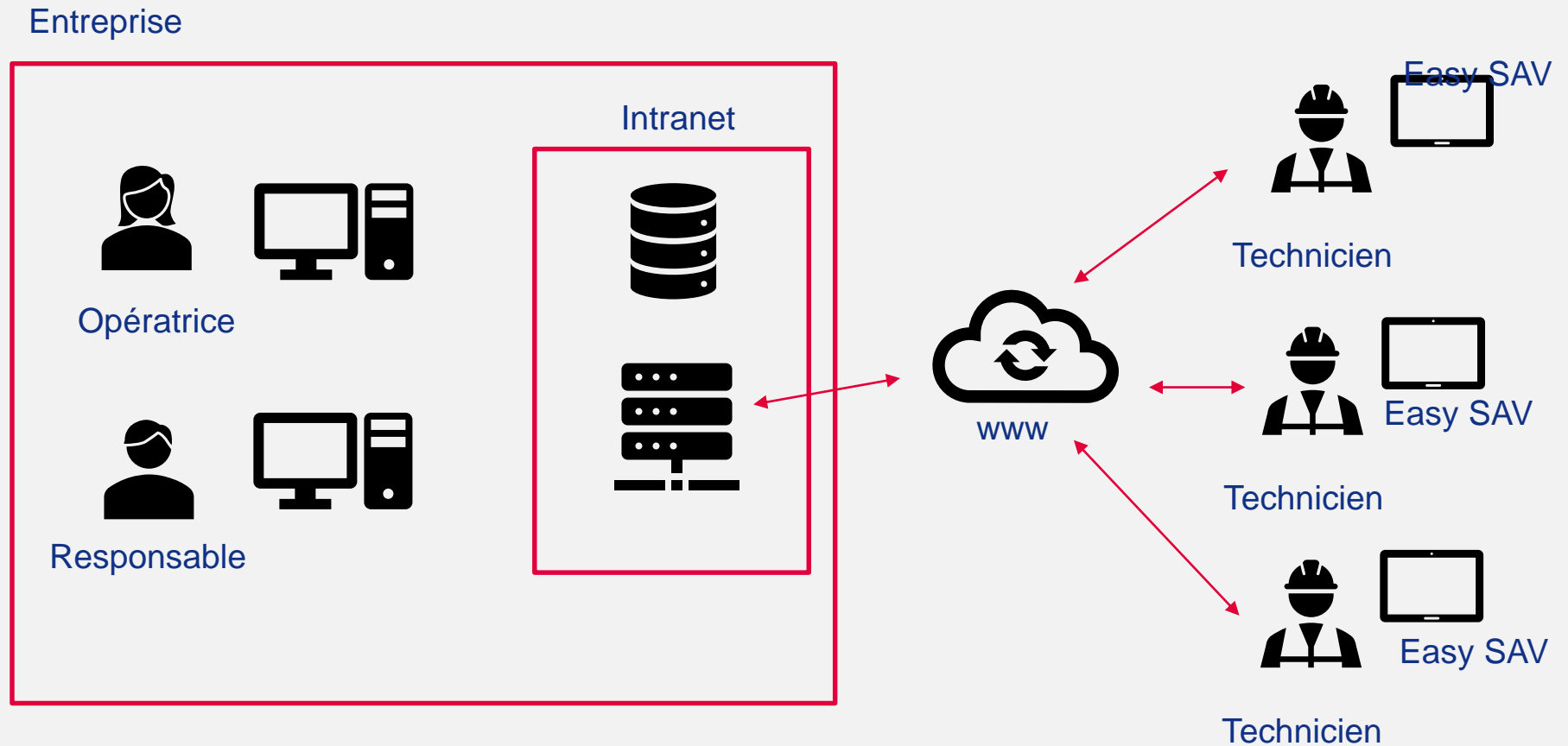
3



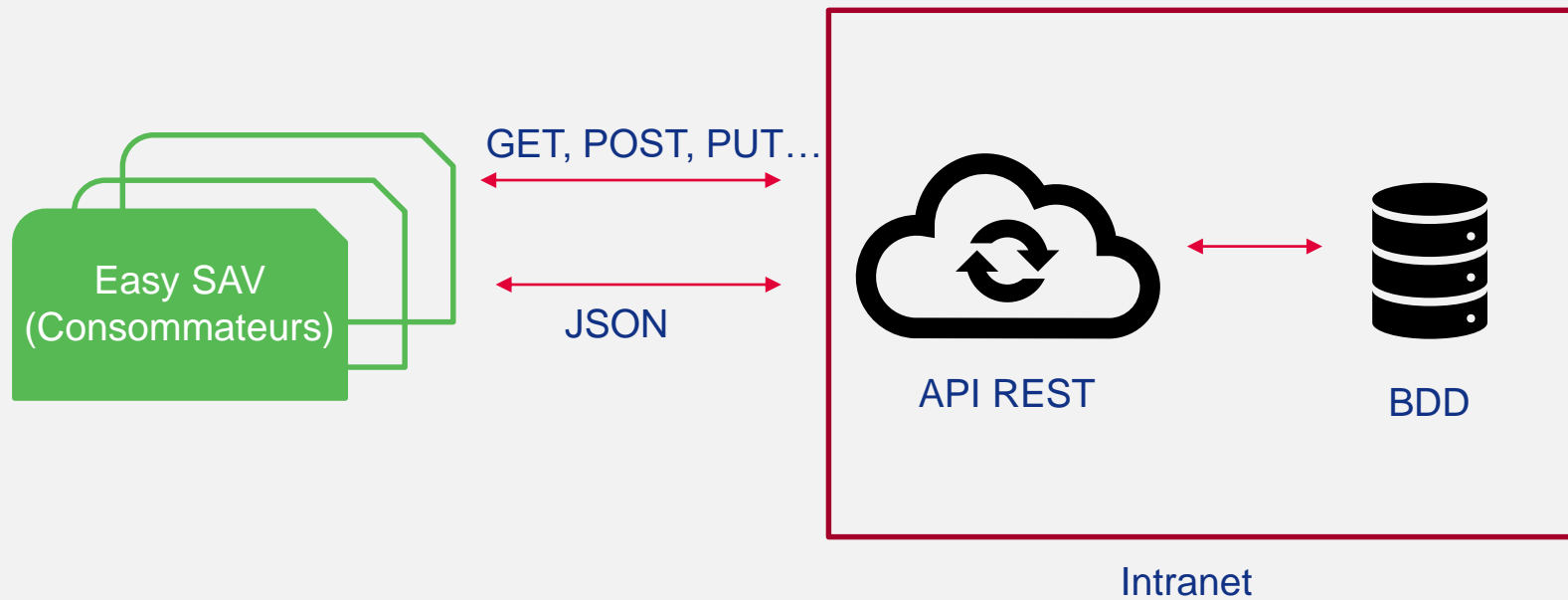
Techniciens

- Télécharger les interventions à effectuer, de la base de données « Intervention »
- Visualiser les interventions de la journée
- Saisir des paramètres techniques afin d'obtenir un diagnostic de panne
- Saisir un compte-rendu d'interventions
- Remonter les interventions effectuées vers le backoffice

EASY SAV - Architecture



EASY SAV - Architecture



REST

- **REST (Representational State Transfer) : Style d'architecture basé sur HTTP permettant la communication au sein des applications Web.**
- **Il s'agit d'un ensemble de conventions et de bonnes pratiques à respecter et non d'une technologie à part entière.**
- **L'architecture REST utilise les spécifications originelles du protocole HTTP, plutôt que d'implémenter une surcouche (comme SOAP).**

Caractéristiques de REST

- Une connexion REST est sans état. Le serveur ne gère pas le contexte de l'échange avec le client.
- URI comme identifiant des ressources
- Verbes HTTP comme identifiant des opérations (GET, PUT, POST, DELETE...)
- Paramètre comme jeton d'authentification

REST : Les Ressources

- Une ressource est un objet identifiable sur le système
Exemple : Contrat, Client, Prêt...
- Une ressource est identifiée par une URI : Une URI
Exemple : `http://bibliotheque/categorie/polar/books/134`



Id du livre
recherché

REST : Les verbes

- Un service REST manipule des entités dénommées ressources via les 4 Verbes http : GET, POST, PUT et DELETE.
- Les verbes POST/GET/PUT/DELETE sont utilisés pour manipuler la collection et/ou les entités, sur le modèle du CRUD.

Verbe HTTP	CRUD	Objectif
POST	Create	Pour créer une entité
GET	Read	Pour obtenir une entité ou une collection d'entités
PUT	Update	Pour mettre à jour une entité
DELETE	Delete	Pour supprimer une entité

Les codes de retour HTTP d'une requête REST sont les suivants :

Code	Erreur
200	Succès
301	Redirection Moved Permanently
404 / 401	Not Found / Unauthorized
500	Erreur serveur Interne Server Error

- **Un service est dit RESTFUL si :**
 - Les méthodes POST/GET/PUT/DELETE sont toutes implémentées.
 - Les méthodes PUT et DELETE sont « idempotentes », ce qui signifie qu'effectuer plusieurs fois la même requête aura le même effet que de l'exécuter une seule fois.
 - La méthode GET est « sûre », ce qui signifie qu'elle ne modifie pas l'état du serveur ni les données (à l'inverse de POST/PUT/DELETE).

- Le modèle d'URI à associer à une ressource est libre. Vous pouvez donc nommer vos ressources comme vous le désirez.
- La méthode la plus répandue consiste à construire le chemin de l'URI en listant les différents niveaux de priorité de la ressource.
- Les éléments de priorité plus importante se trouveront plus à gauche (début) de l'URI. Par exemple, nous pouvons avoir un service qui gère les contrats par domaine.
 - Exemples :
<http://monserveur/axa/vie/contrats>
<http://monserveur/axa/iard/contrats/contrat/1205255>

REST : Construction et sémantique

- Le modèle d'URI à associer à une ressource est libre. Vous pouvez donc nommer vos ressources comme vous le désirez.
- La méthode la plus répandue consiste à construire le chemin de l'URI en listant les différents niveaux de priorité de la ressource.
- Les éléments de priorité plus importante se trouveront plus à gauche (début) de l'URI. Par exemple, nous pouvons avoir un service qui gère les contrats par domaine.
 - Exemples :
<http://monserveur/axa/vie/contrats>
<http://monserveur/axa/iard/contrats/contrat/1205255>

Sémantique	Méthode	Requête HTTP
Récupérer un contrat	getContrat	GET /contrats/contrat/123457
Modifier un contrat	updateContrat	PUT /contrats/contrat/123
Récupérer tous les contrats	getContrats	GET /contrats/

Représentation des ressources REST : Le format JSON

- **JSON est l'acronyme de JavaScript Object Notation. C'est un format texte qui permet de représenter des données et de les échanger facilement au même titre qu'XML.**
- **Encodage d'une requête REST**
 - On utilise typiquement un en-tête HTTP
 - **Content-Type: application/json**
 - Le client doit demander un format d'encodage spécifique en utilisant l'entête
 - **Accept: application/json**
- **Exemple d'une remontée d'une collection d'objets de type WorkItem :**

```
[{"Id":10,"TypeCarte":"User Story","Etat":"En cours","Titre":"Mon élément de travail  
10","Designation":"azerty","DateEntree":"24/08/2012","DateSortie":null,"Persona":null},  
{"Id":20,"TypeCarte":"Bug","Etat":"En cours","Titre":"Mon élément de travail","Designation":"qsdfghjjkk  
20","DateEntree":"24/08/2012","DateSortie":null,"Persona":null}]
```
- **L'avantage majeur de JSON est de pouvoir facilement intégrer ces objets dans une application cliente utilisant du JavaScript.**

Utiliser FLASK

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

@app.route('/')
def hello():
    return 'Hello, World!'

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

- * Serving Flask app "classement_flask" (lazy loading)
- * Environment: production
WARNING: Do not use the development server in a production environment.
Use a production WSGI server instead.
- * Debug mode: off
- * Running on **http://127.0.0.1:5000/** (Press CTRL+C to quit)

■ Implémentation de la route

```
@app.route('/api/classement/')
def classement():
    dico_L1 = {
        'titre': 'Classement Ligue 1',
        'classement' : {'PSG':58, 'OM':49, 'Rennes':41, 'LOSC':40}
    }
    return jsonify(dico_L1)
```

■ Test unitaire

```
def test_classement(self):
    response = self.app.get('/api/classement/', follow_redirects = False)
    s = json.loads(response.data)
    self.assertEqual(response.status_code, 200)
    self.assertEqual(s["titre"], "Classement Ligue 1")
```

■ Implémentation de la route

```
@app.route("/api/classement/add", methods=['POST'])
def add():
    new_team = request.get_json(force=True)
    if "Equipe" in new_team:
        equipe = new_team["Equipe"]
    if "Points" in new_team:
        points = new_team["Points"]
    return ("OK", {"Content-Type": "application/plaintext"})
```

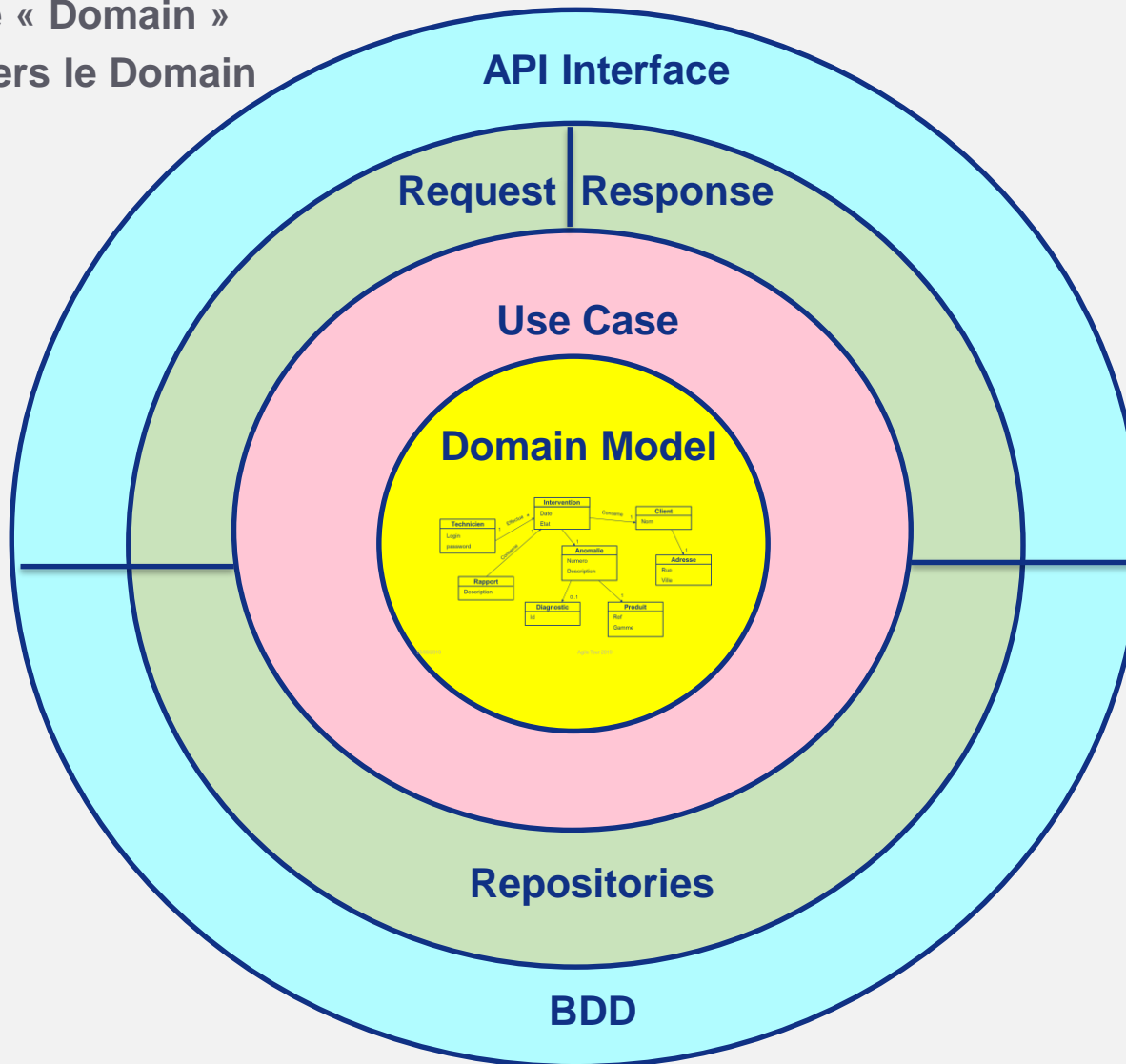
■ Test unitaire

```
def test_add_root(self):
    response = self.app.post('/api/classement/add', data=
        json.dumps({'Equipe':'OL', 'points': '34'}),
        content_type='application/json')
    self.assertEqual(response.status_code, 200)
```

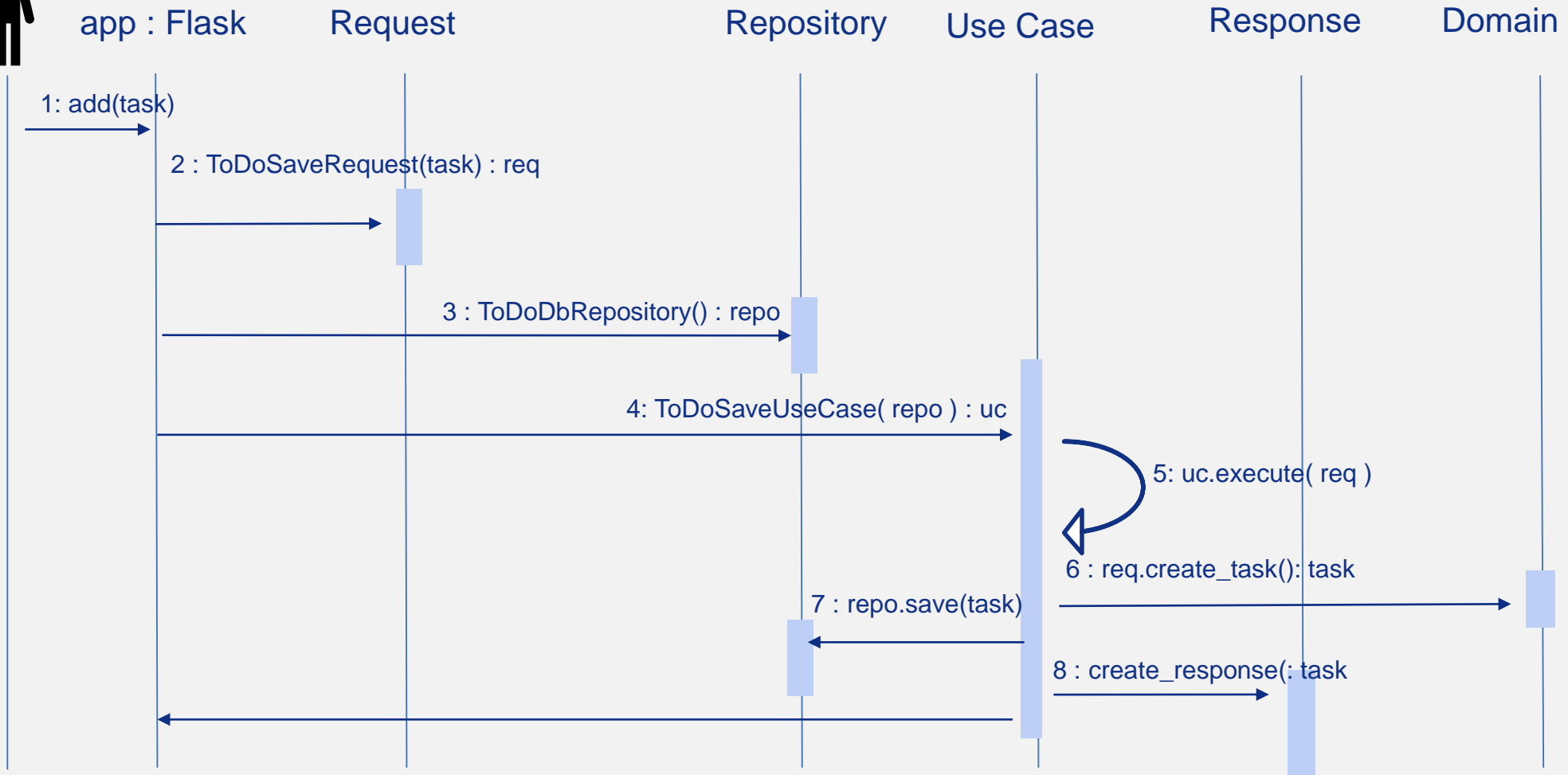
- **Clean Architecture apporte une réponse aux contraintes des développements en Agile :**
 - Un découplage entre les tierces parties et le cœur de métier,
 - La mise en place d'un contexte favorable à la pratique du TDD et du BDD.
 - Une flexibilité face aux choix techniques (indépendance de l'UI et BDD)
 - Une plus grande tolérance au changement

CLEAN ARCHITECTURE

- Modèle orienté « Domain »
- Dépendance vers le Domain



CLEAN ARCHITECTURE



Structure du projet

