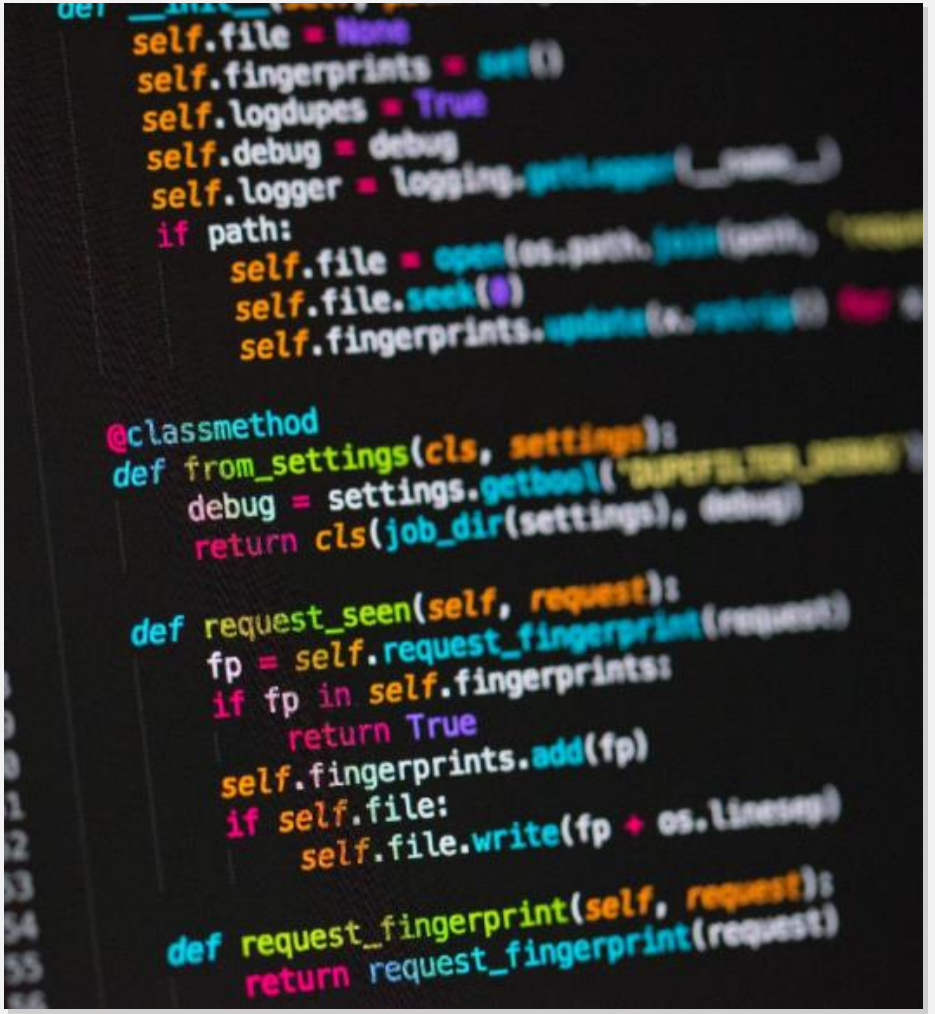
# Programmation PYTHON

EPSI – Module 3ème année

## Objectifs de la formation

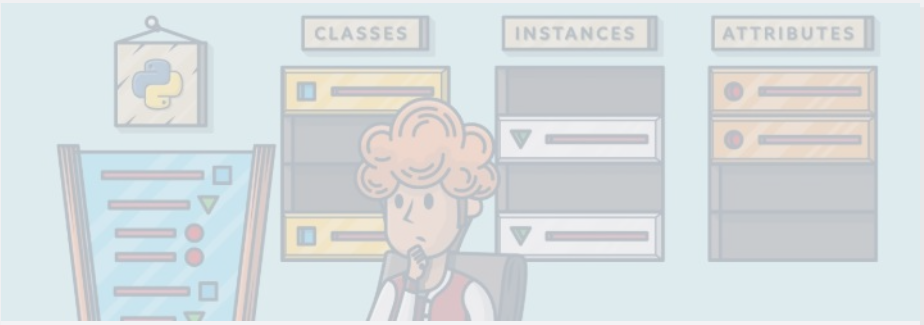
### OBJECTIFS

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

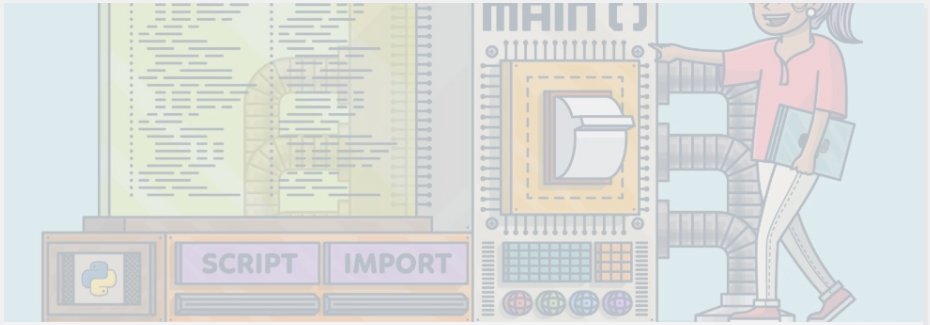


## Sommaire

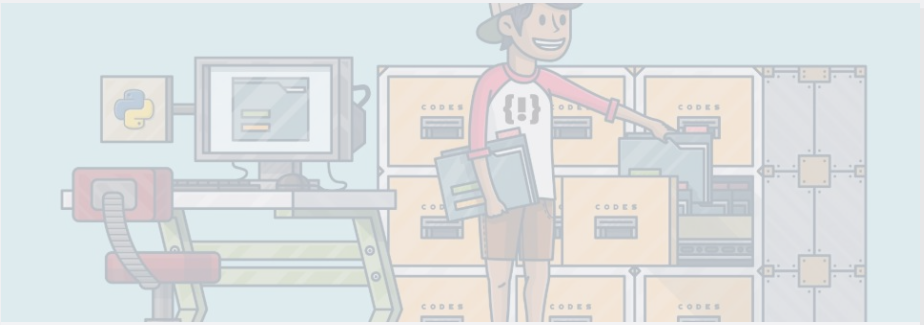
Les bases du langage




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/**

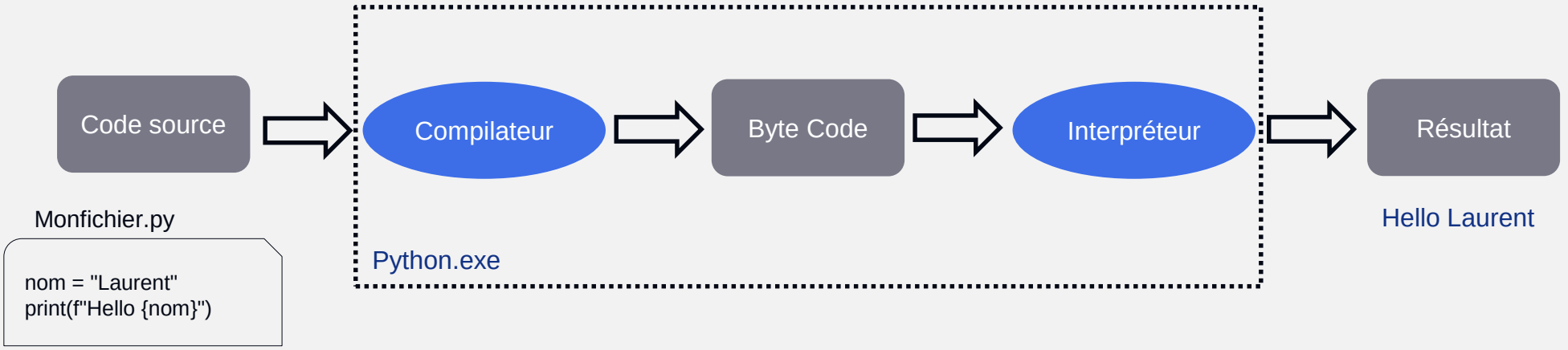
## Présentation des caractéristiques du  langage

- 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
  - Rapsberry, 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 | float | complexe | bool |
|---|---|---|---|
| 0 78 -5 | 5.8 -3.9 1.45e-7 | 5 + 7j | True False |

- Type itérable

| str | list | tuple | set |
|---|---|---|---|
| "anom" ‘tel’ | [1,5,6] ["a", "b"] | (1,5,6) | {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, réference 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
```

## Booléen et None

cours_csharp = **False**

int(cours_csharp) == 0

cours_python = **True**

int(cours_ python) == 1

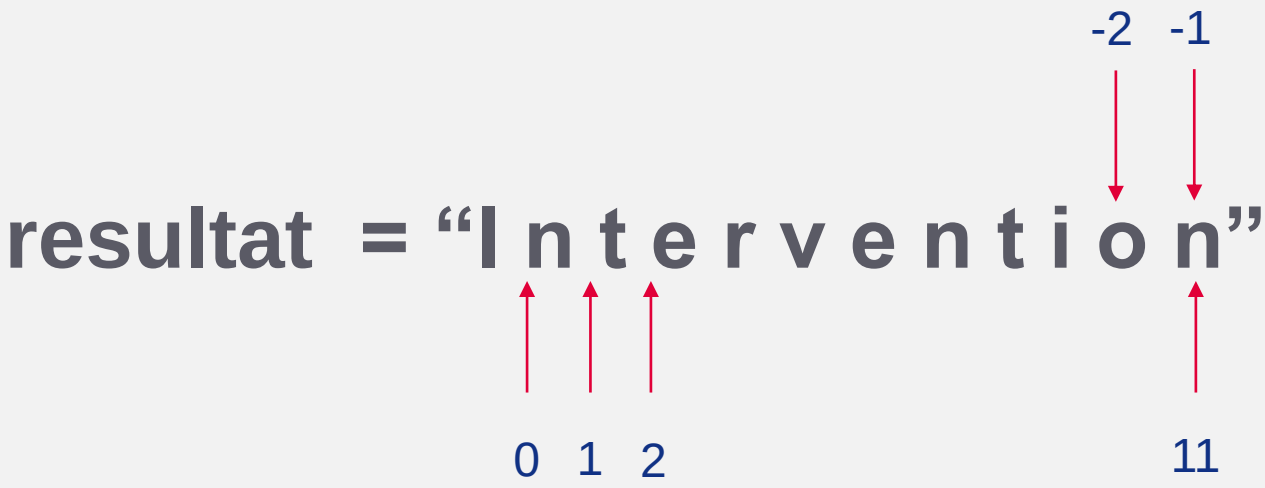
alimente_cours = **None**

## 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"[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   (invertion des lettres en commençant par la fin
```

## Programme console

```
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/lnyffels/Code_Epsi/blob/master/Exercices/ennonce.md

## if-else

```
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
```

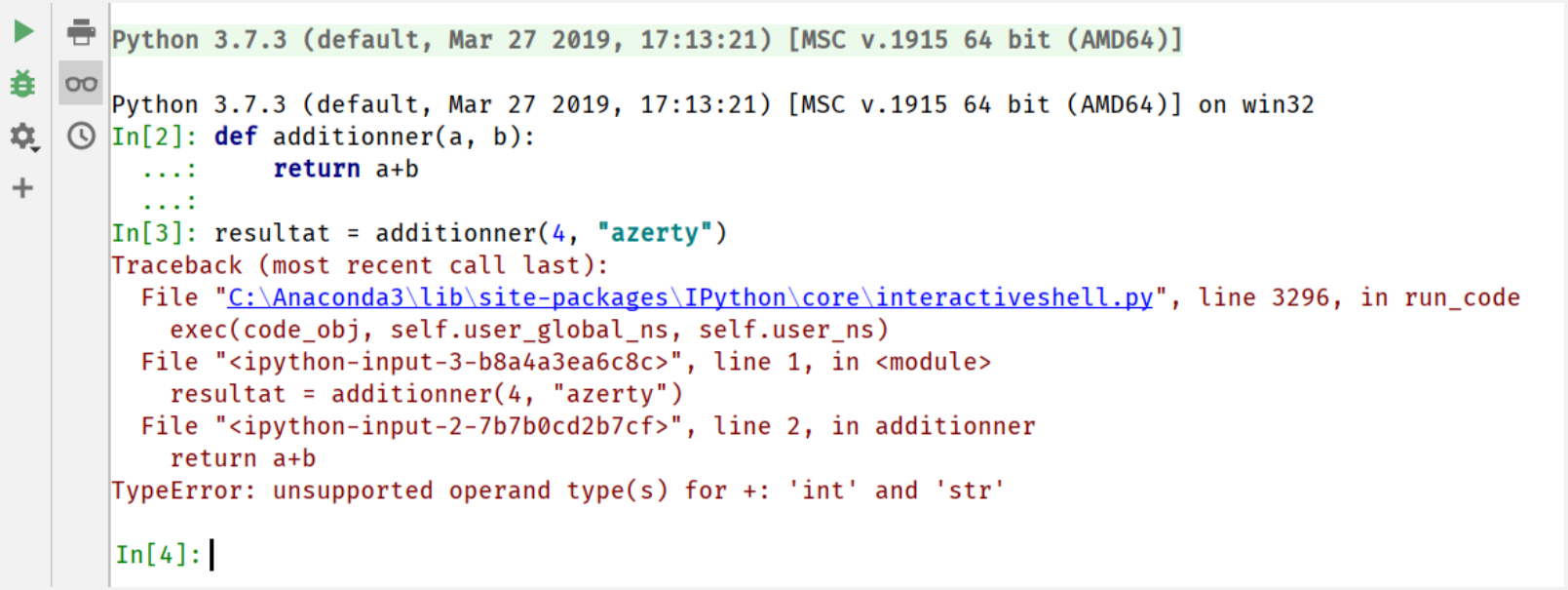
## Fonction – retour multiple

```
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 chaine 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
lst = []
lstNumber = [10, 20, 30, 40]
lstNumberType: List[int] = [3, 6, 8, 34, 10, 567]
lstHybrid = [34, 'azerty', (4, 8), [1, 2, 3]]
listInitNumber = range(30)
lstCinqZeros = [0]*5

# Accès au premier élément
elem = lstNumberType[0]

# Concaténation
lstTotal = lstNumber + lstNumberType

# Copie
lstCopie = lstNumberType
```

## Liste

```
lg = len(lstNumberType)                          # taille de la liste
lstNumberType.append(7)                          # ajoute un élément à la fin de la liste
lstNumberType.sort()                             # trie la liste
lstNumberType.reverse()                          # inverse la liste
positionElem = lstNumberType.index(7)            # recherche l'élément 7 dans la liste
lstNumberType.remove(10)                         # retire un élement de la liste
elem = lstNumberType.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**

## Comprehension de liste

- 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 compréhesion
  - [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 chaine 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()
dicoInit = {"nom": "Bob", "age": 42, "identifiant": "S611456"}

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

# copie de dictionnaire
dicoCopie = dicoInit

# 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)
```

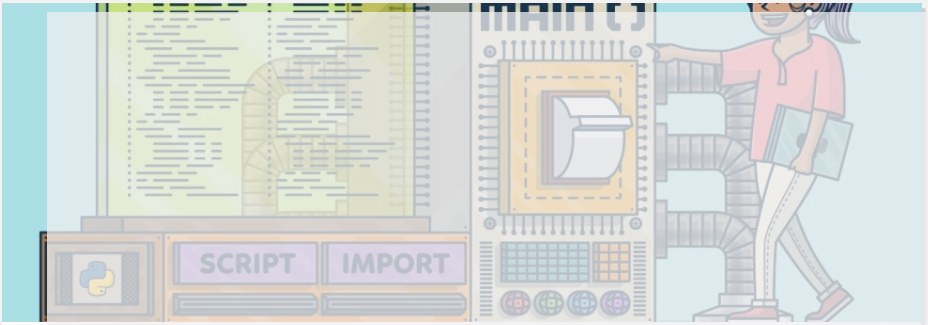
## Fonction qui retourne un objet de type fonction

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

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

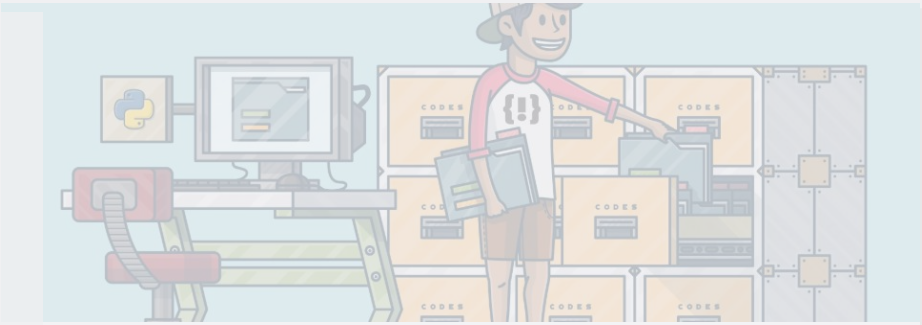
## Sommaire

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("lnf")
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 abstraite 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
```

# *TP*

- Gestion de compte bancaire

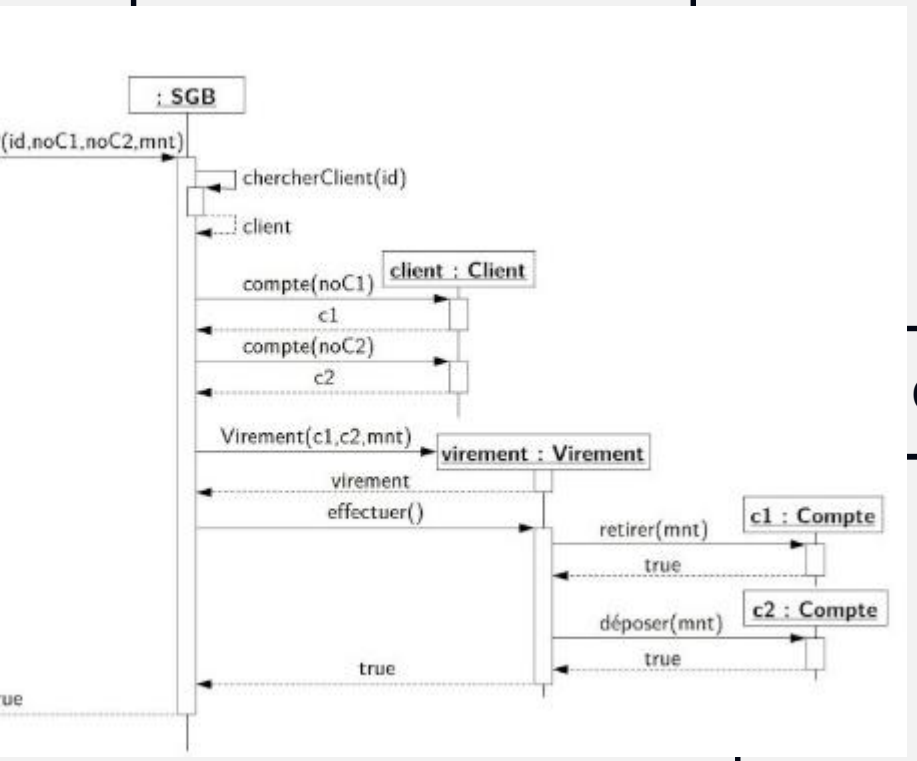
# Scenarios GHERKIN

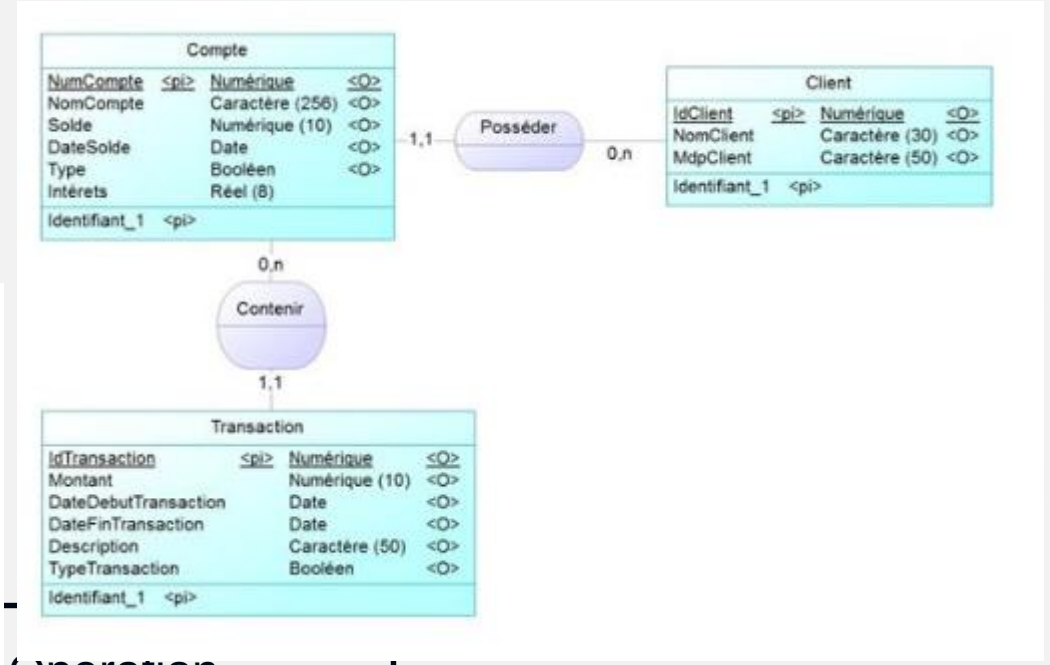
```
Feature: Crediter_compte
  """
  EN TANT QUE client de la banque
  JE PEUX créditer mon compte courant
  AFIN DE l'alimenter
  """

  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

Client

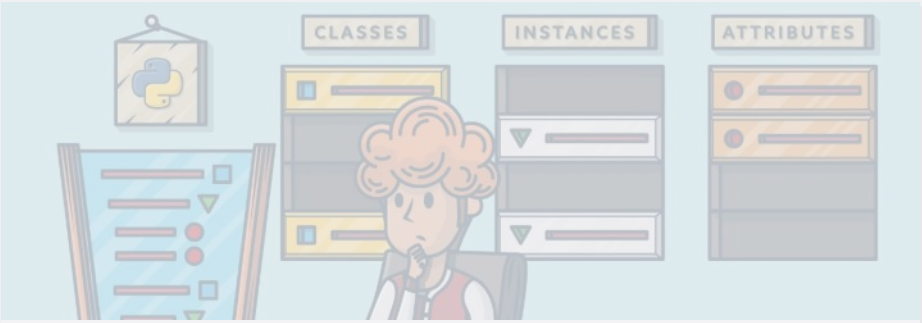


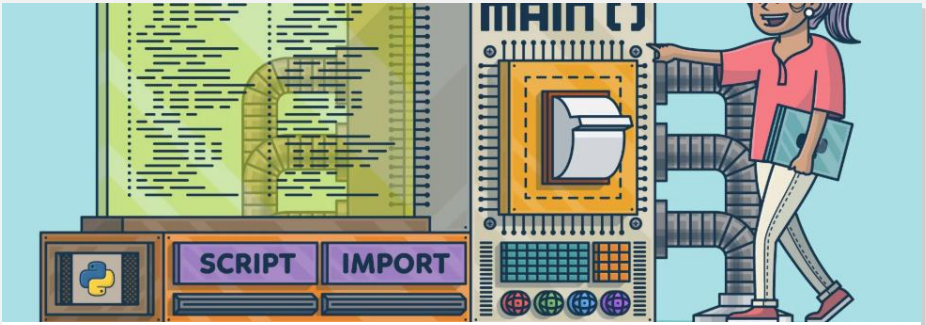

Operation

## Sommaire

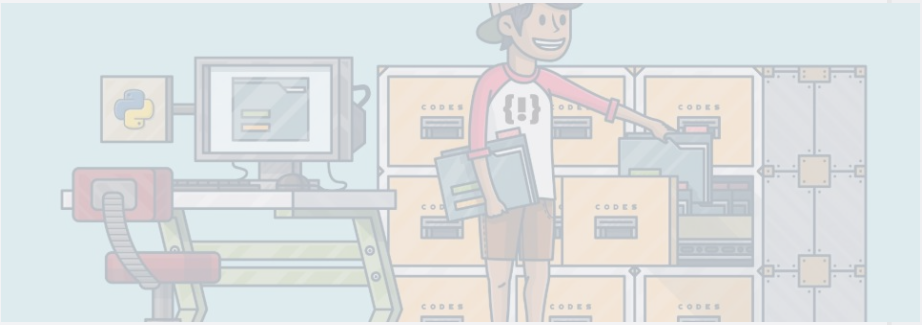
Les bases du langage




Programmation Objet en Python




Utiliser les bibliothèques standards



Structurer, concevoir et refactoriser une application

## Quelques bibliothèques standard

- Fichier Texte
- Fichier Json
- BDD sqlite3
- tkinter
- SQLAlchemy

## Lecture / Ecriture dans un fichier

```
import os
os.chdir("c:/temp")

mon_fichier = open("monfichier.txt", "w")
mon_fichier.write("ecrire une premiere ligne")
mon_fichier.close()

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

# *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

## 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"]
}
# Récupére 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
- Lègère : 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…)**

```
cursor.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**

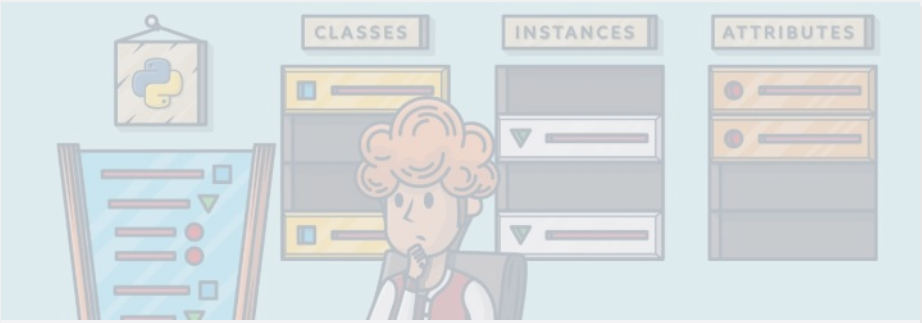
```
curseur.execute("SELECT valeur FROM maTable WHERE data = ?", donnee)
print(curseur.fetchone())   # Retourne un tuple


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

- **Fermer une connexion**

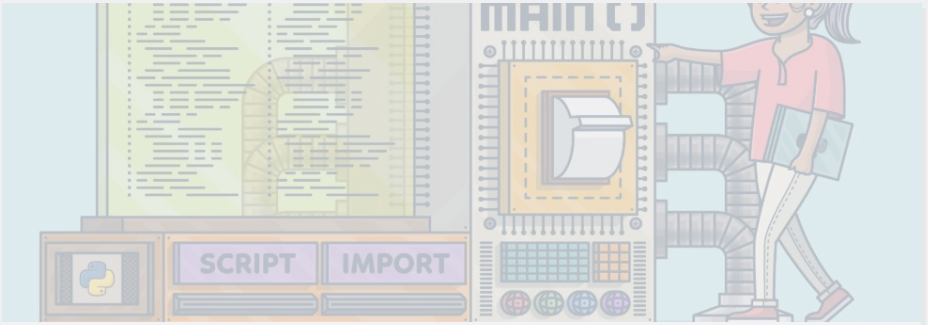
```
connexion.close()
```

## Sommaire

Les bases du langage




Programmation Objet en Python




Utiliser les bibliothèques standards


**Structurer, concevoir, refactoriser une application**

# Clean Code



VS

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

## TDD – Test Driven Development

- 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"

# Démonstration Transfert compte courant

## Excercice 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%.

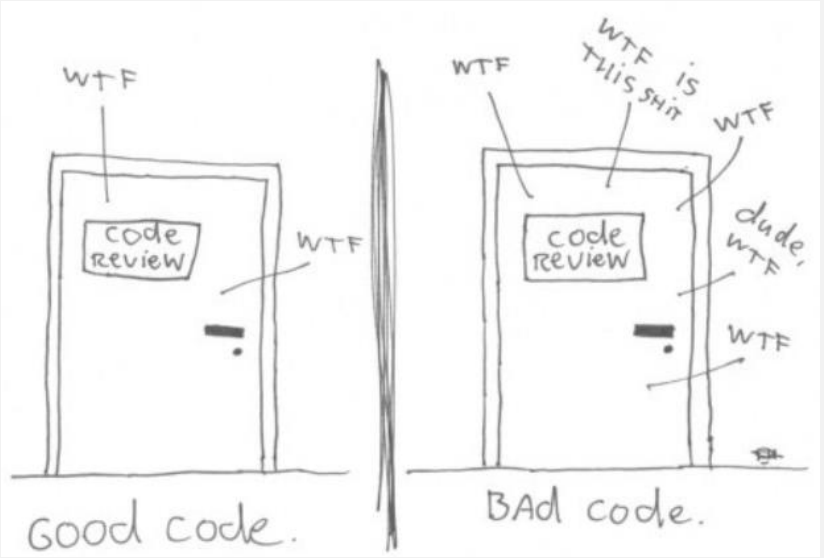
## 2. Ne contient aucune duplication

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

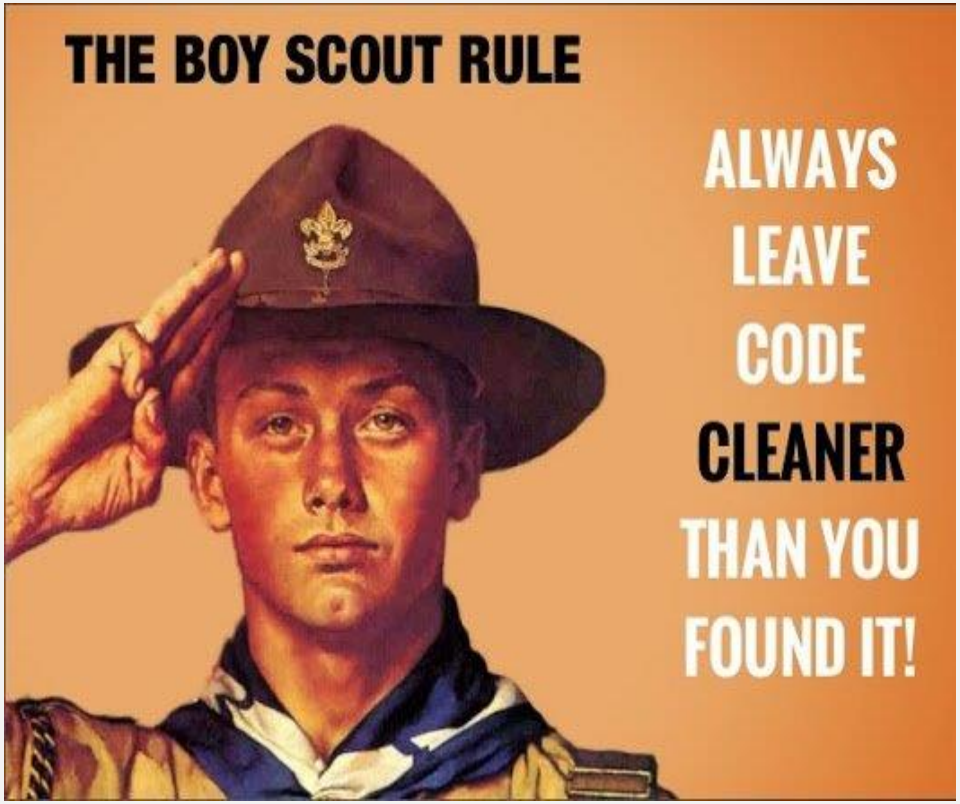
## 3. Minimum en nombres de classes, en longueur de méthodes…

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

## 4. Exprime l’intention



# La « Boy Scout Rule »



- 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 |

# Long Method

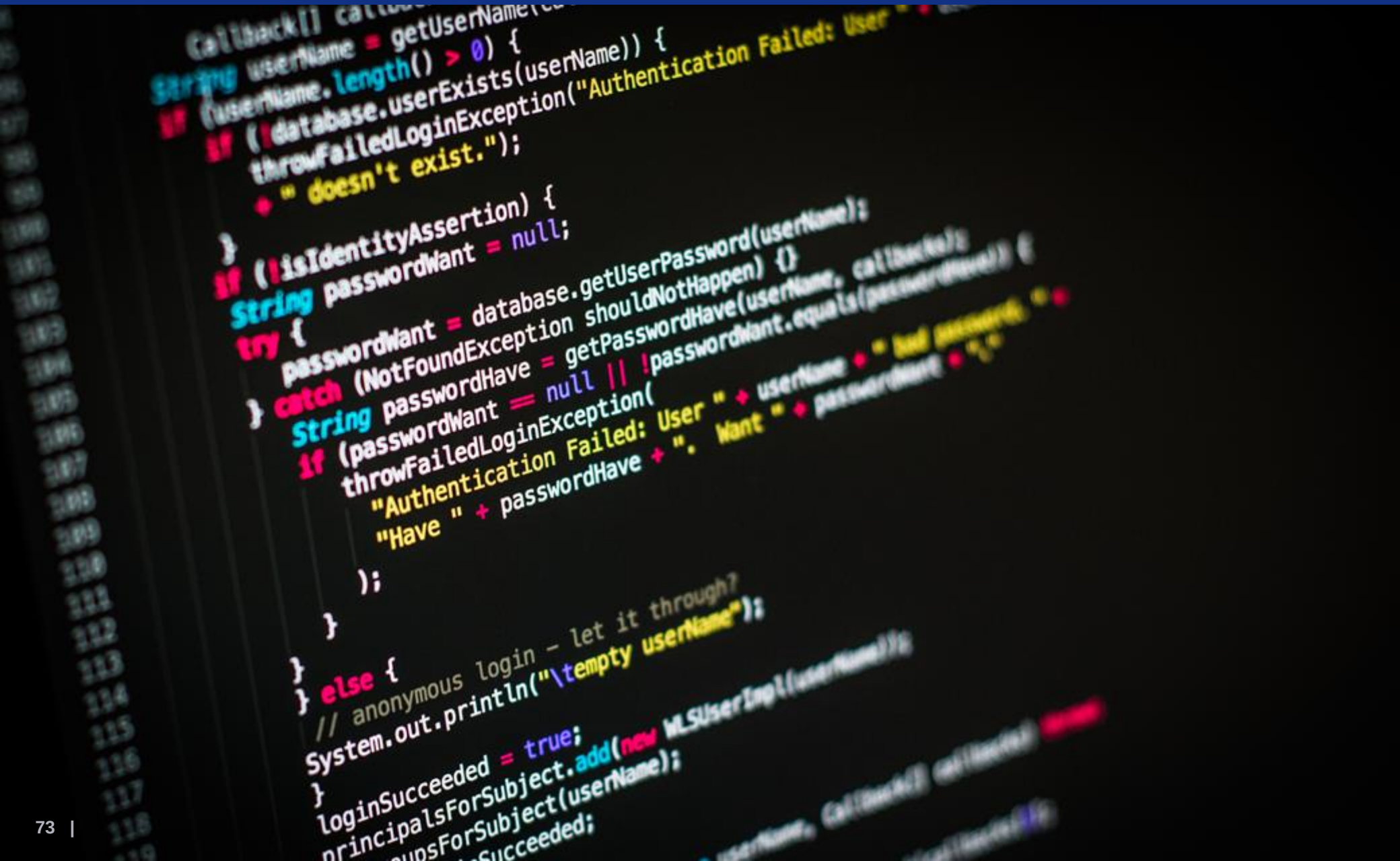
## Raison du problème

- 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

# Extract Method

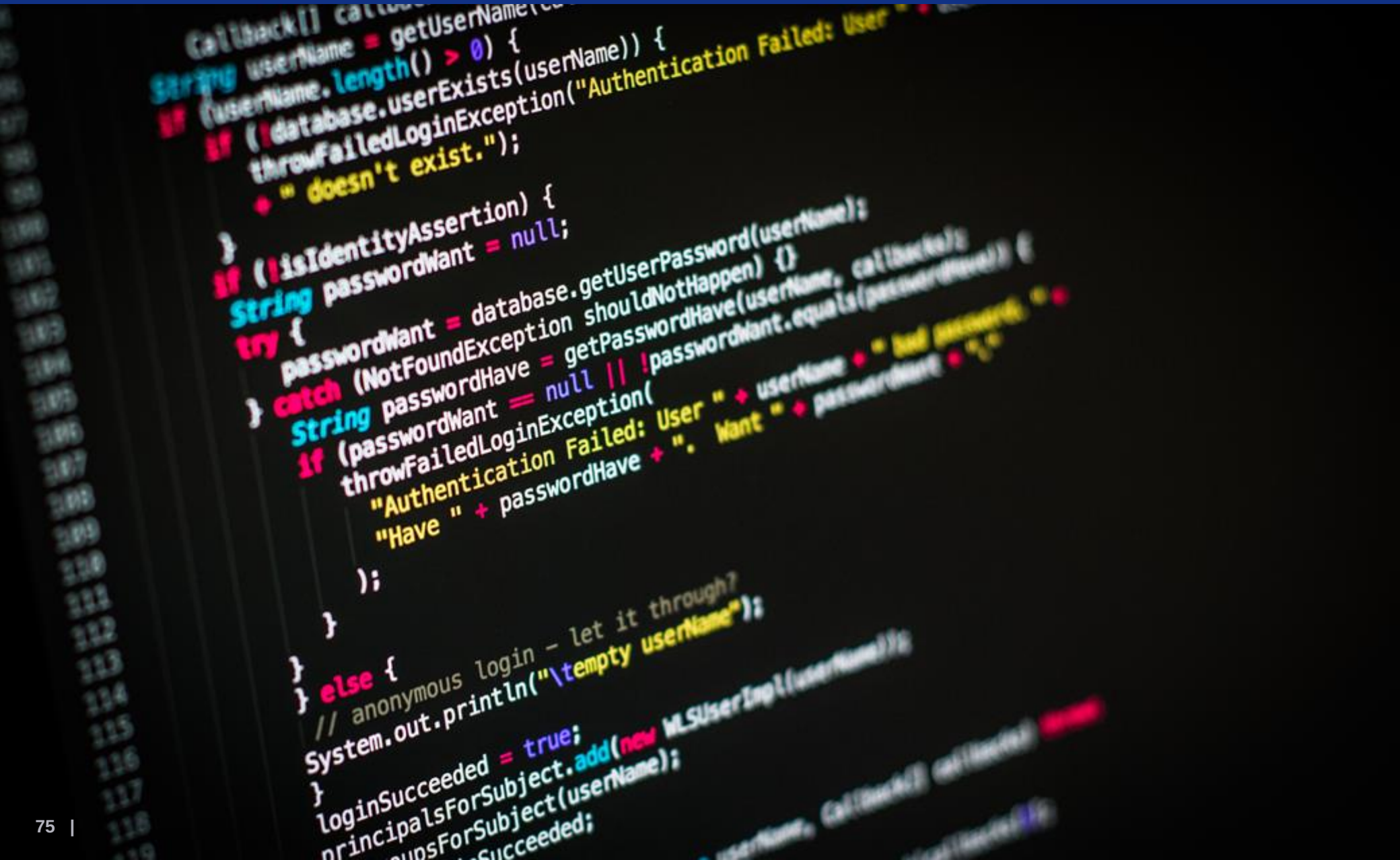


# Extract Method

## 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)
```

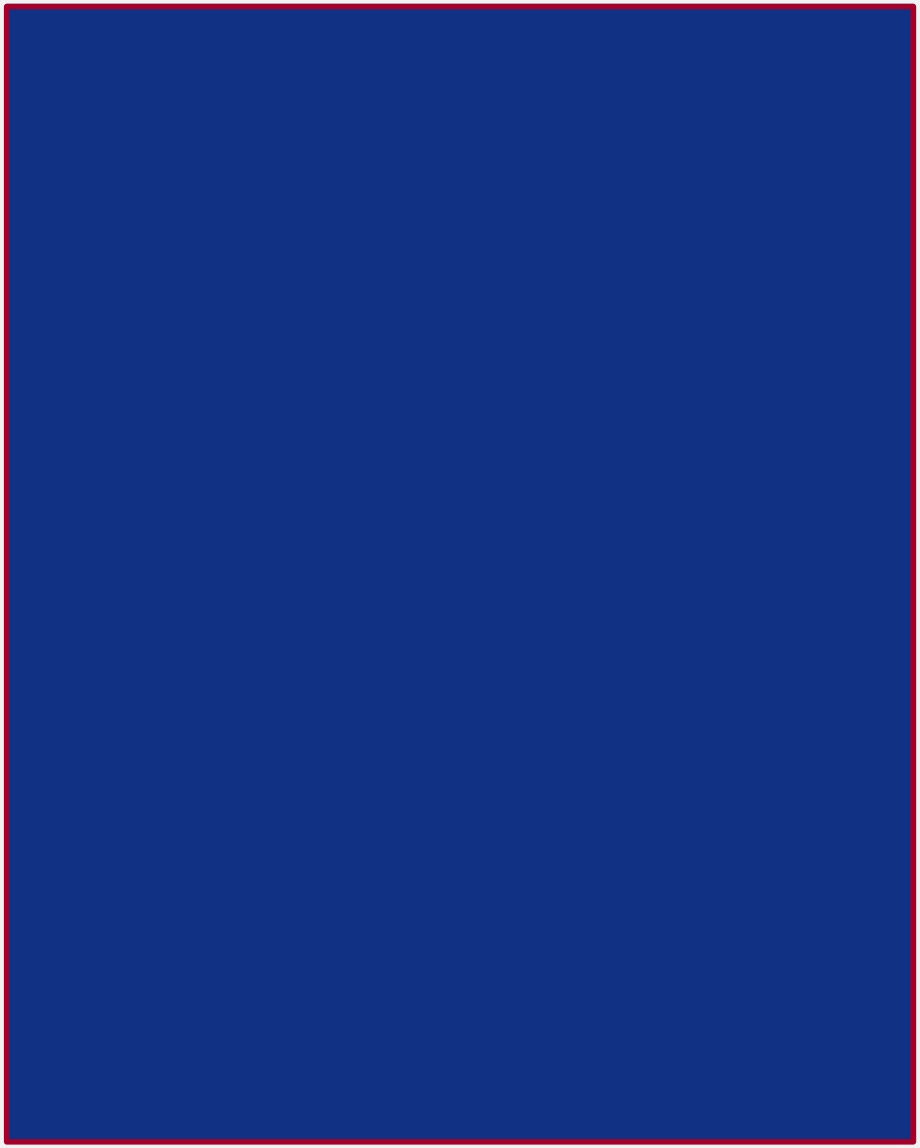
# Replace Temp with Query



# Replace Temp with Query

## Code smell : **long method**

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



# Replace Temp with Query

## Problème :

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

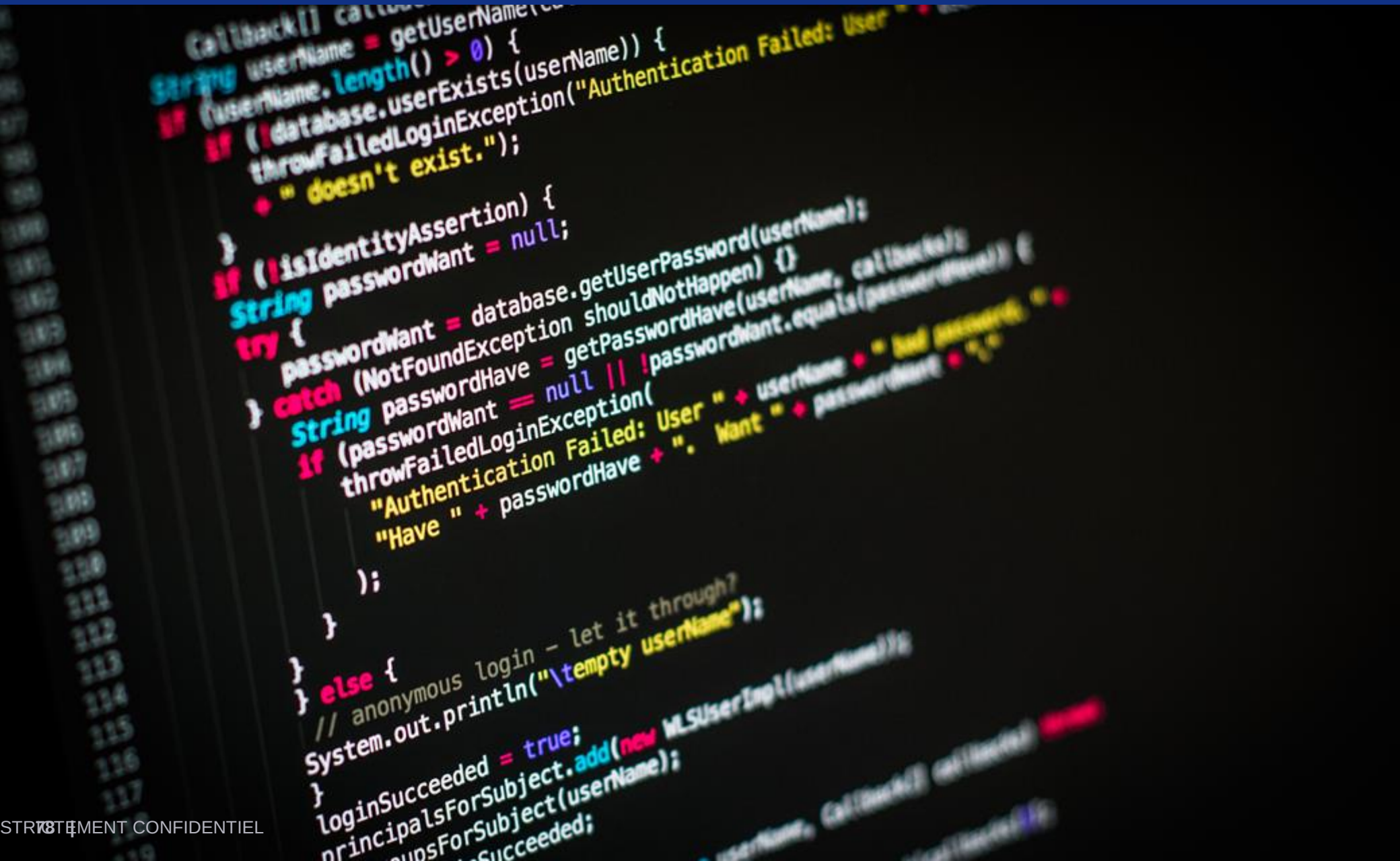
## Solution :

- 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



# 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
```

# Introduce Parameter Object

## 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.

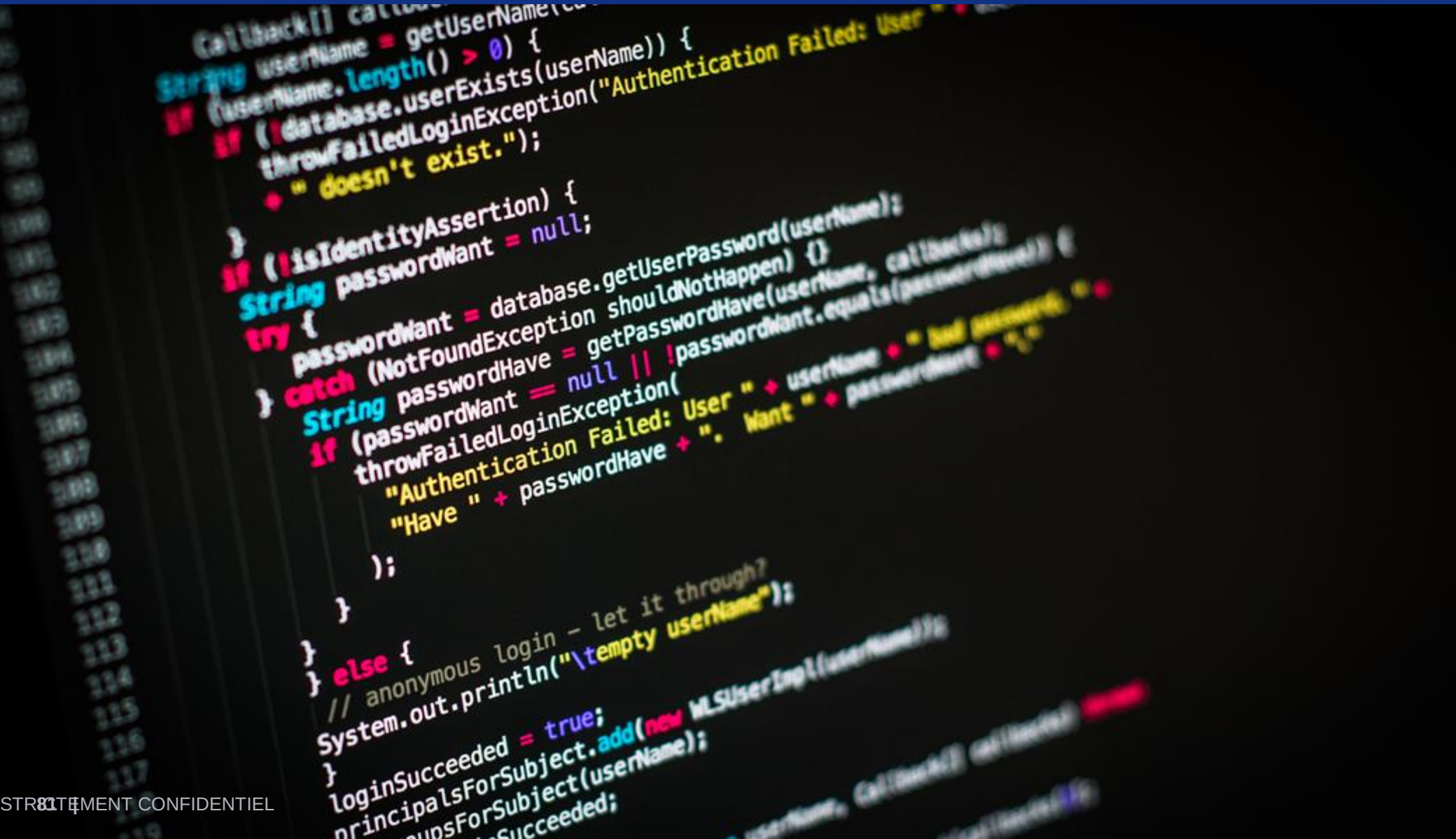
## Avantage:

- 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.

# Preserve Whole Object

réinventons / notre métier

## Démo



# Preserve Whole Object



## 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
```

# Preserve Whole Object



## 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.

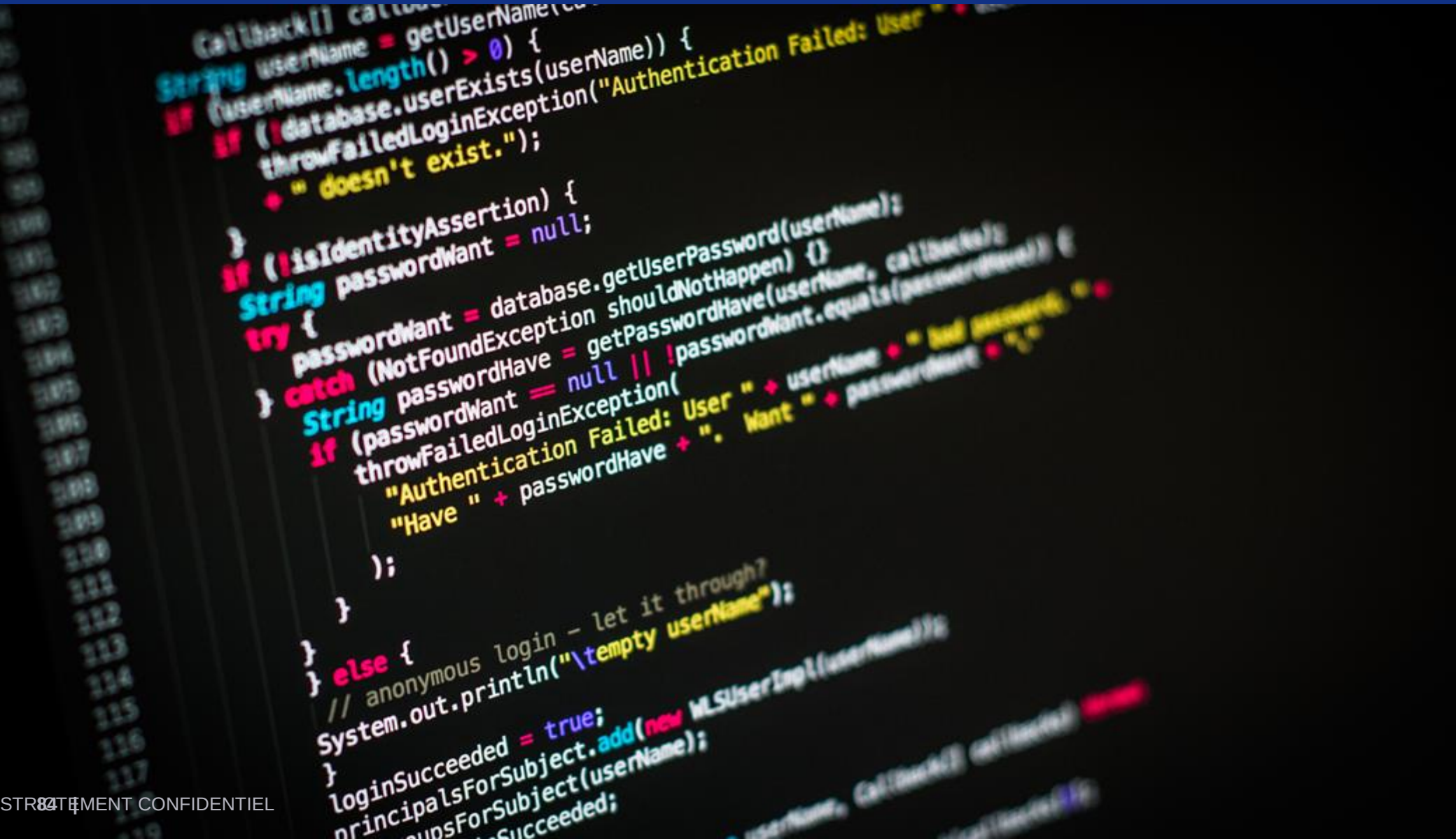
## Avantage:

- 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.

# Decompose Conditional



## Démo



# Decompose Conditional



## Code Smell

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

# Decompose Conditional

## 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 ».

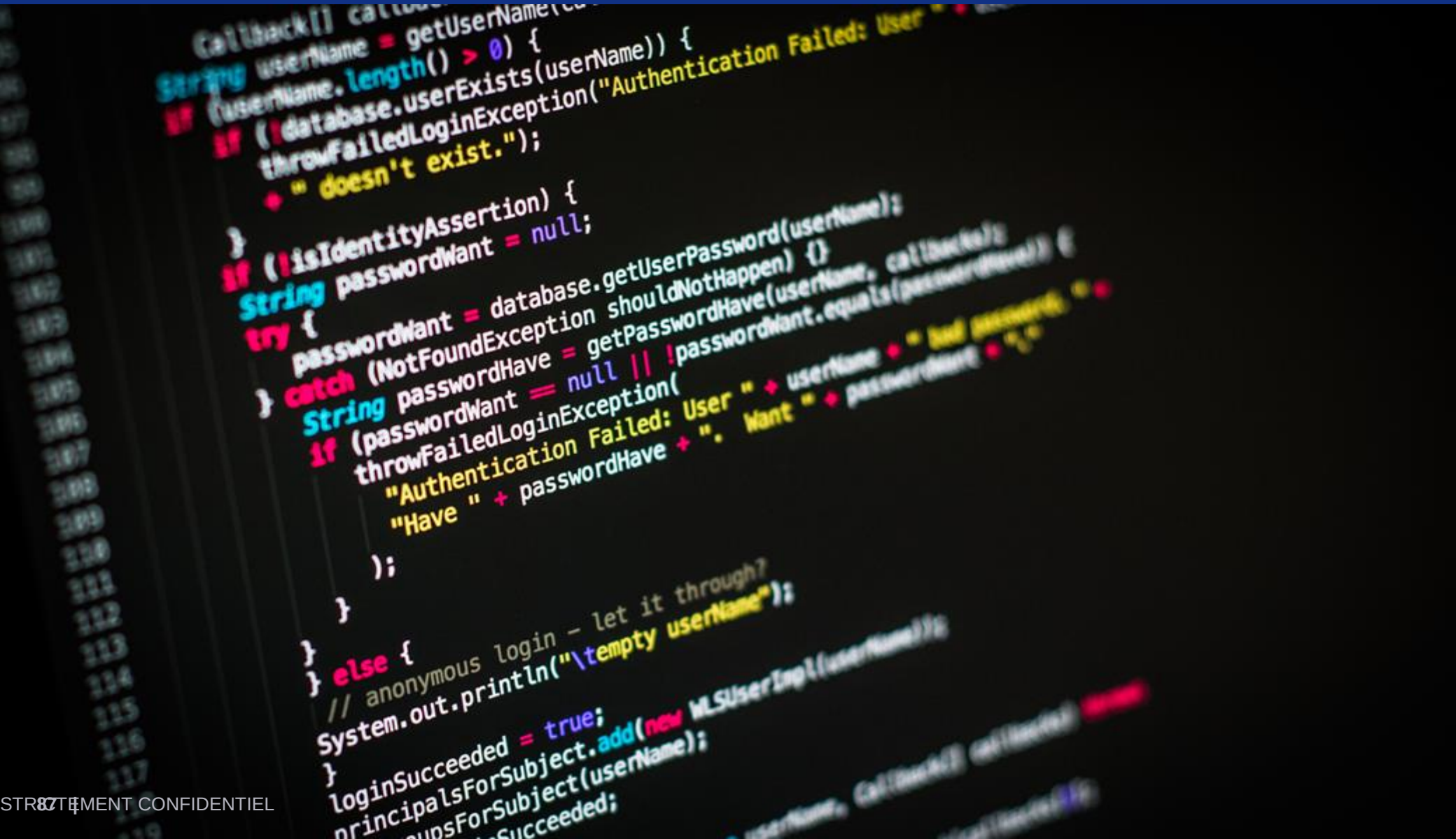
## Avantage:

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

# Replace Method with Method Object

réinventons / notre métier

## Démo



# Replace Method with Method Object



## 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
```

# Replace Method with Method Object

## 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.

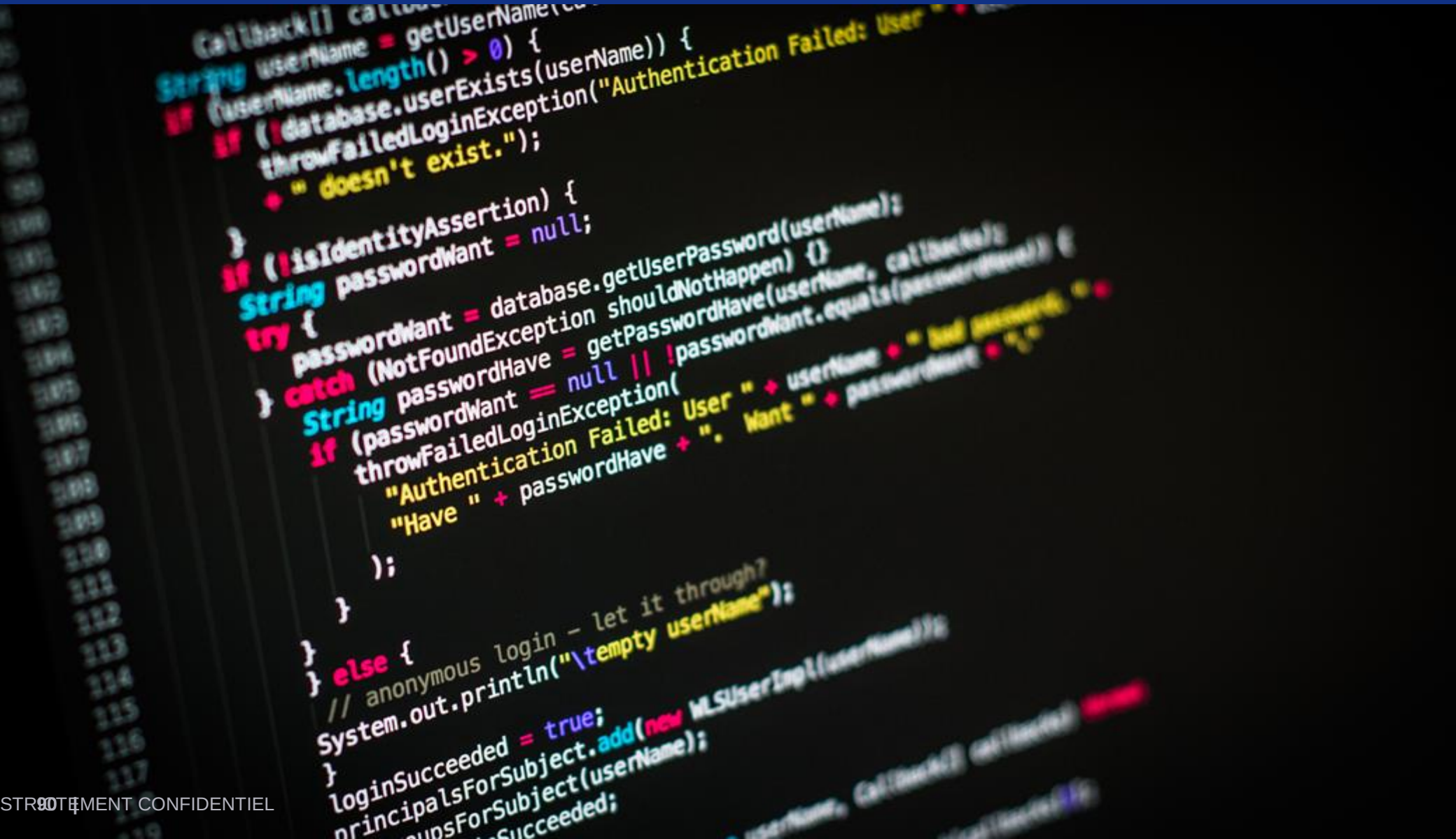
## Avantage:

- 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.

# Extract Subclass

réinventons / notre métier

## Démo



# Large Class



## Raison du problème

## Techniques disponibles

- Extract Class / Subclass / interface
- Duplicate Observed Data

# Extract Subclass

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



## 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
```

# Extract Subclass



## 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.

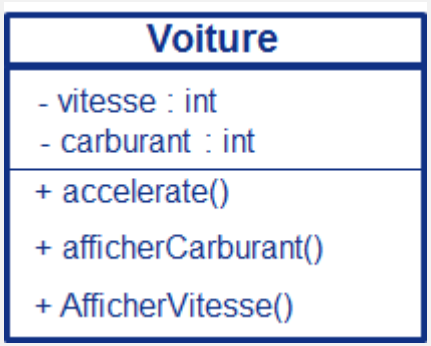
## Avantage:

- Respect des principes SOLID.

# Duplicate Observed Data



## Code Smell



# Duplicate Observed Data



## 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:

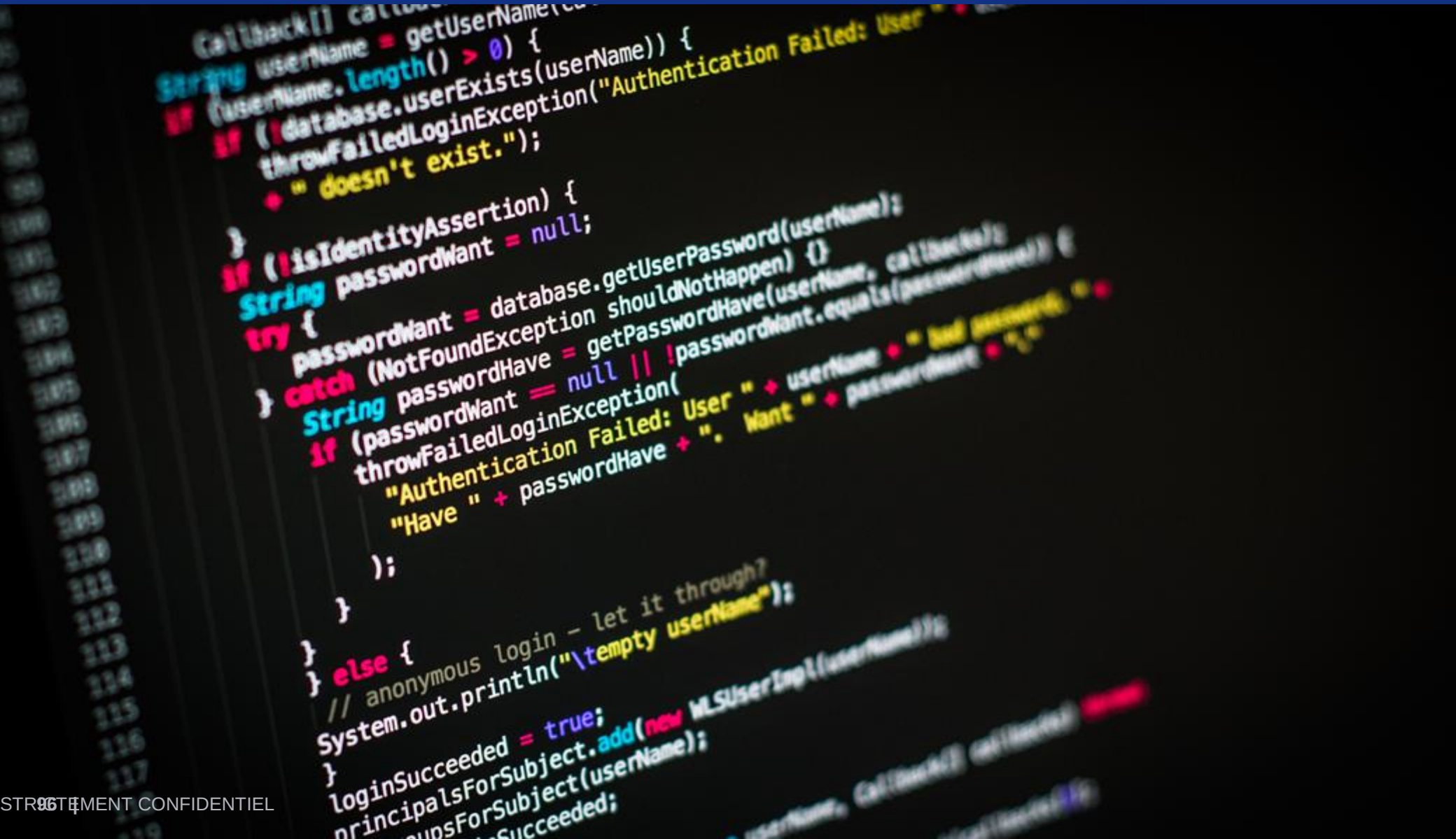
- Séparation des responsabilités entre les classes qui traitent de la logique métier et celles qui portent l'affichage des données (Single Responsability Principle)

- 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)

# Duplicate Code

réinventons / notre métier

## Démo



# Duplicate code



## Raison du problème

## Techniques disponibles

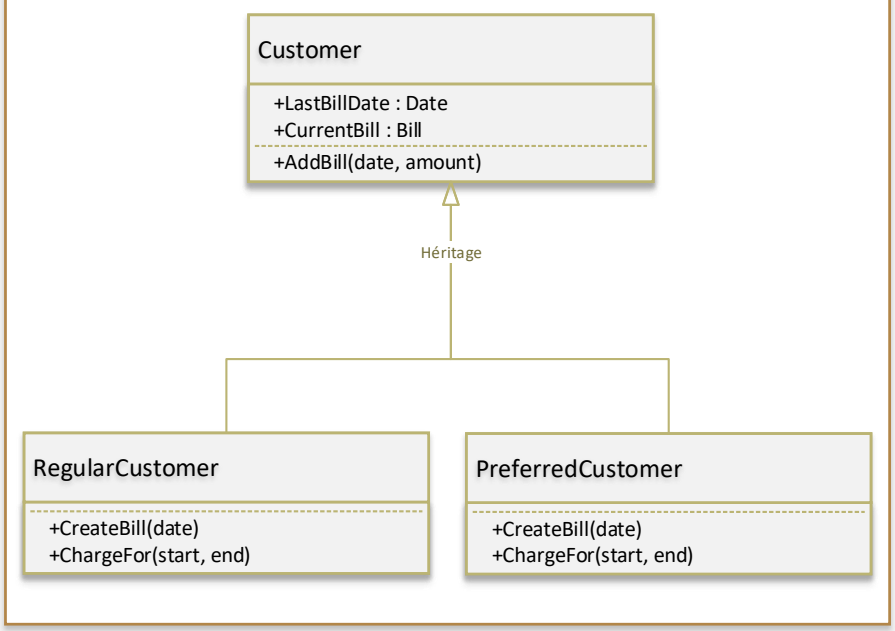
- Extract Method (déjà vu)
- Pull Up Method
- Form Template Method

# Pull Up Method



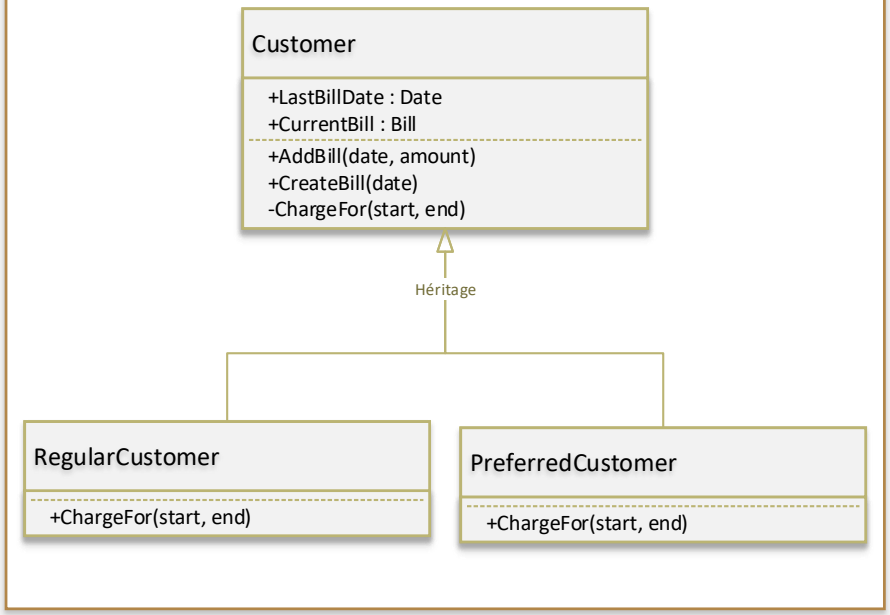
## Code Smell

Problème




## Solution possible

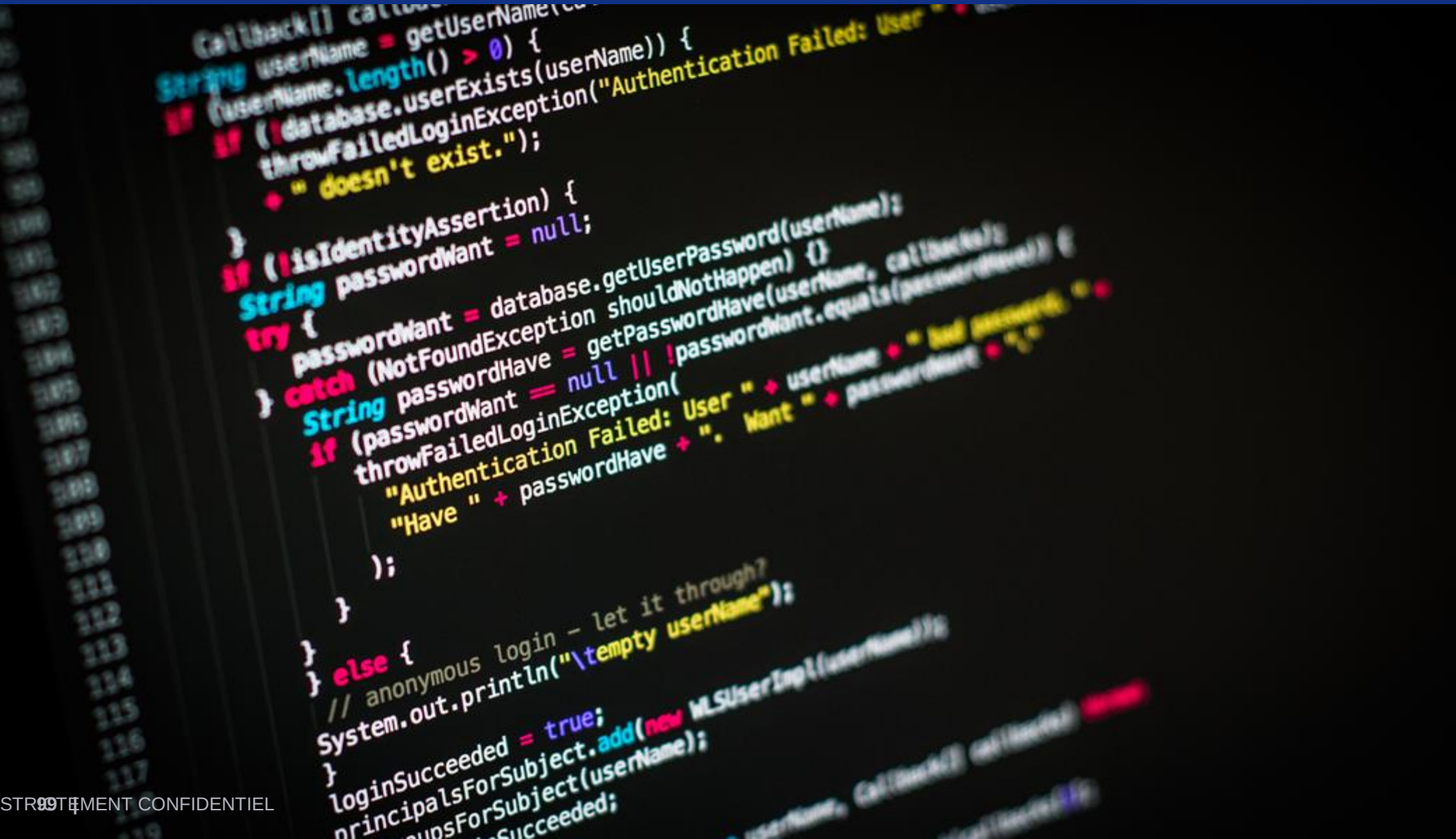
Solution possible



# Pull Up Method

réinventons / notre métier

## Démo



# Pull Up Method



## Code Smell

```
class Customer:
    def __init__(self):
        self.lastBillDate = None
        self.currentBill = None

    def addBill(self, date, amount):
        self.currentBill = Bill(date, amount)


class RegularCustomer(Customer):
    def __init__(self, dateLastBill):
        self.lastBillDate = dateLastBill

    def createBill(self, date):
        self.addBill(date, self.chargeFor(self.lastBillDate, date))

    def chargeFor(self, endDate, startDate):
        return (endDate - startDate).days * 100


class PreferredCustomer(Customer):
    def __init__(self, dateLastBill):
        self.lastBillDate = dateLastBill

    def createBill(self, date):
        self.addBill(date, self.chargeFor(self.lastBillDate, date))

    def chargeFor(self, endDate, startDate):
        return (endDate - startDate).days * 80
```

# Pull Up Method

## Explication:

- Vos sous-classes ont des méthodes qui effectuent un travail similaire
- Rendez les méthodes identiques, puis déplacez-les vers la superclasse concernée.

## Raison du problème:

- Les sous-classes ont grandi et se sont développées indépendamment les unes des autres, provoquant des champs et des méthodes identiques (ou presque identiques)

## Avantage:

- Se débarrasse du code en double. Si vous devez apporter des modifications à une méthode, il est préférable de le faire au même endroit que de rechercher tous les doublons de la méthode dans des sous-classes.
- Cette technique de refactoring peut également être utilisée si, pour une raison quelconque, une sous-classe redéfinit une méthode de superclasse mais effectue essentiellement le même travail.

# Naming smell



## 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.

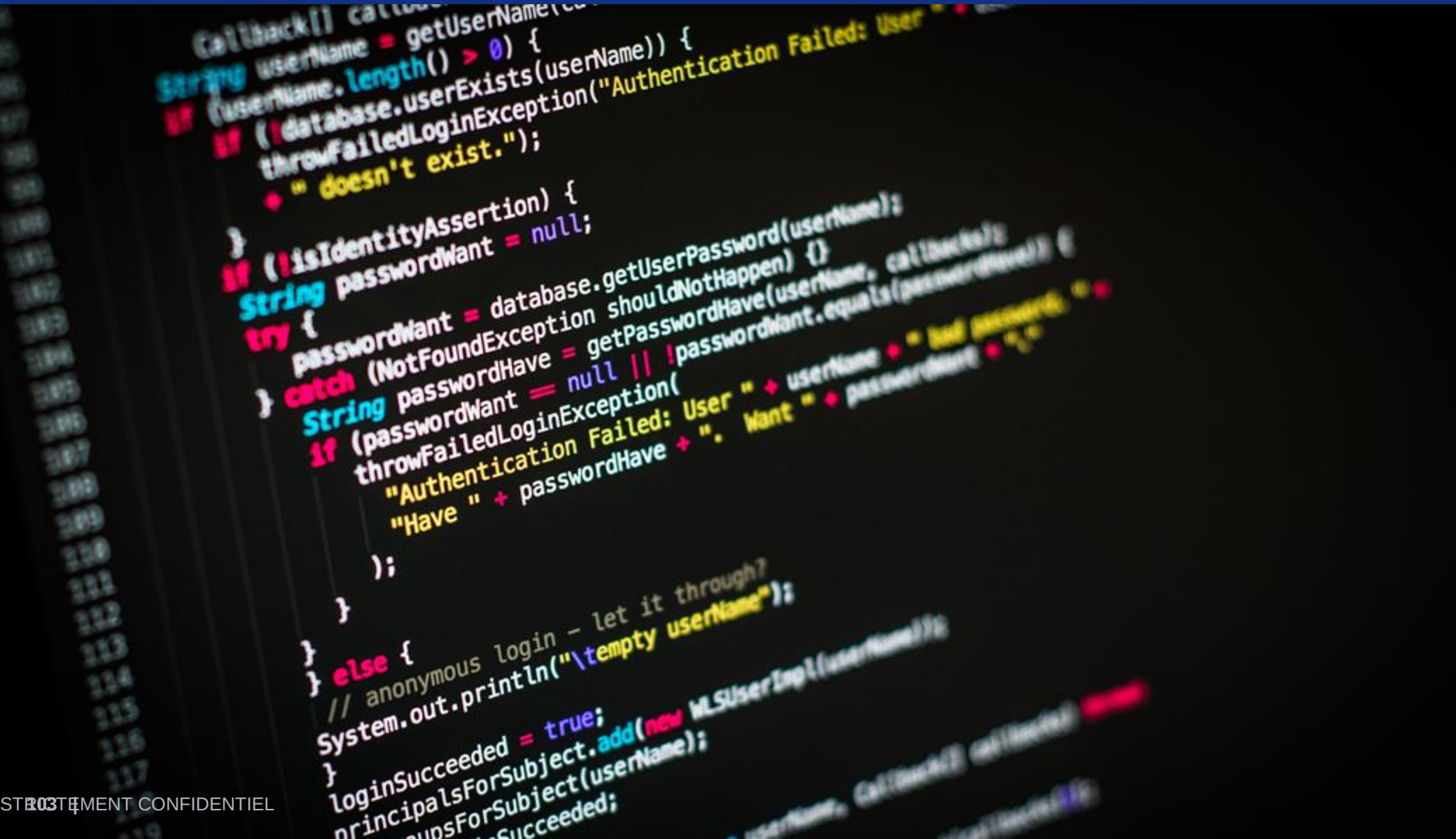
## Techniques disponibles

- Remove Double Negation

# Remove Double Negative

réinventons / notre métier

## Démo



# Remove Double Negative



## 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
```

# Remove Double Negative



## 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 entraine 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.

# Couplers



## Raison du problème

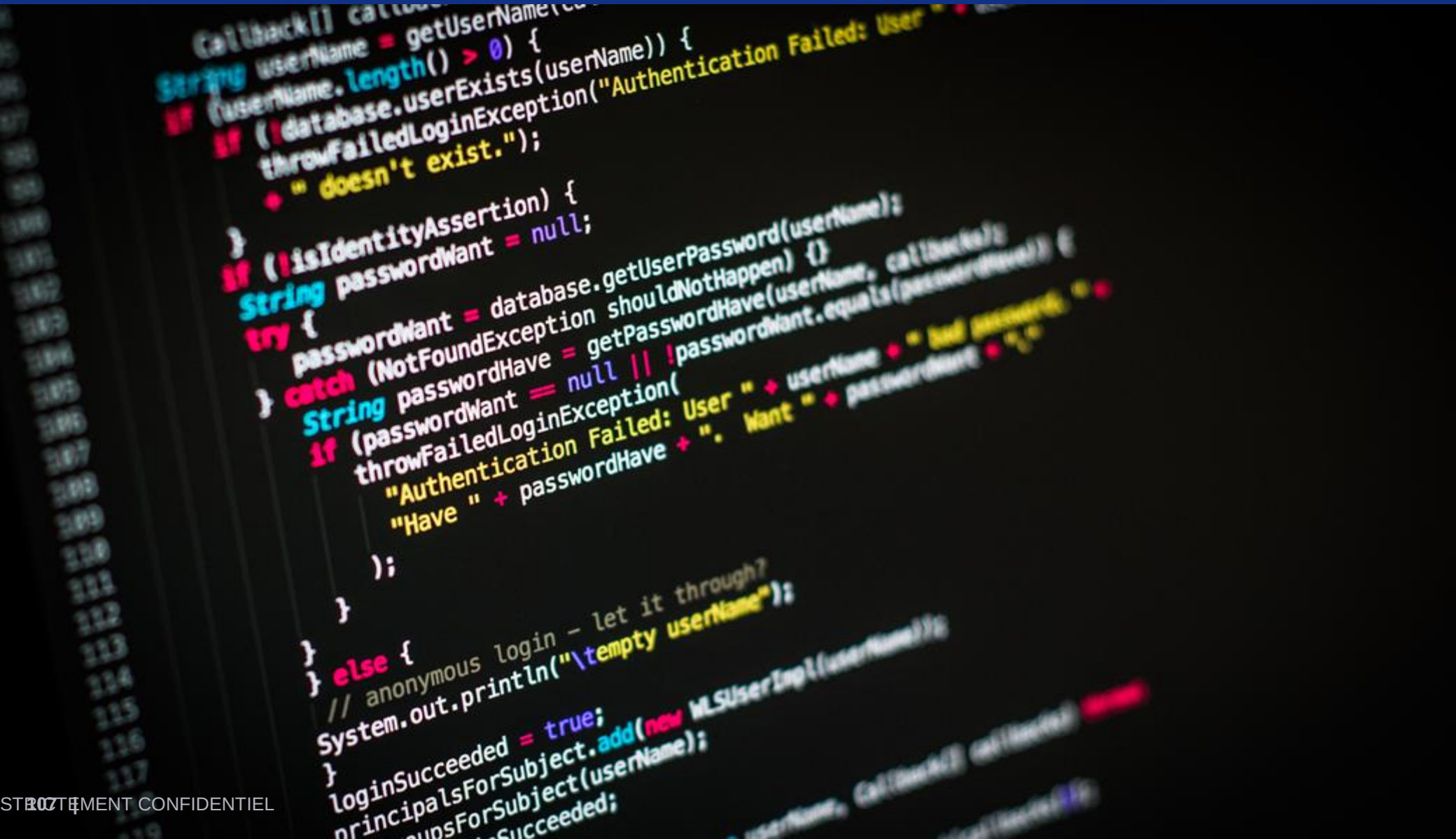
## Techniques disponibles

- Introduce Parameter Object
- Decompose Conditional
- Message Chains
- Middle Man

# Feature Envy

réinventons / notre métier

## Démo



# Feature Envy



## Code Smell

```
class TripService:
    def getTripsByUser(self, user):
        tripList = []
        loggedUser = UserSession.getInstance().getLoggedUser()
        isFriend = False
        if loggedUser is not None:
            for friend in user.getFriends:
                if friend.equals(loggedUser):
                    isFriend = True
                    break
            if isFriend:
                tripList = TripDAO.findTripsByUser(user)
            return tripList
        else:
            raise UserNotLoggedException()
```

## Solution possible

lo

# Feature Envy

## Explication:

- Une méthode accède aux données d'un autre objet plus qu'à ces propres données. (la méthode est « envieuse » des « fonctionnalités » de l'autre objet)

## Raison du problème:

- Ce code smell peut se présenter si l'on ne respecte pas les responsabilités des classes que l'on utilise.
- Il peut aussi arriver dans le cas ou des propriétés ont été déplacées dans une classe mais pas le comportement liés à ces propriétés.

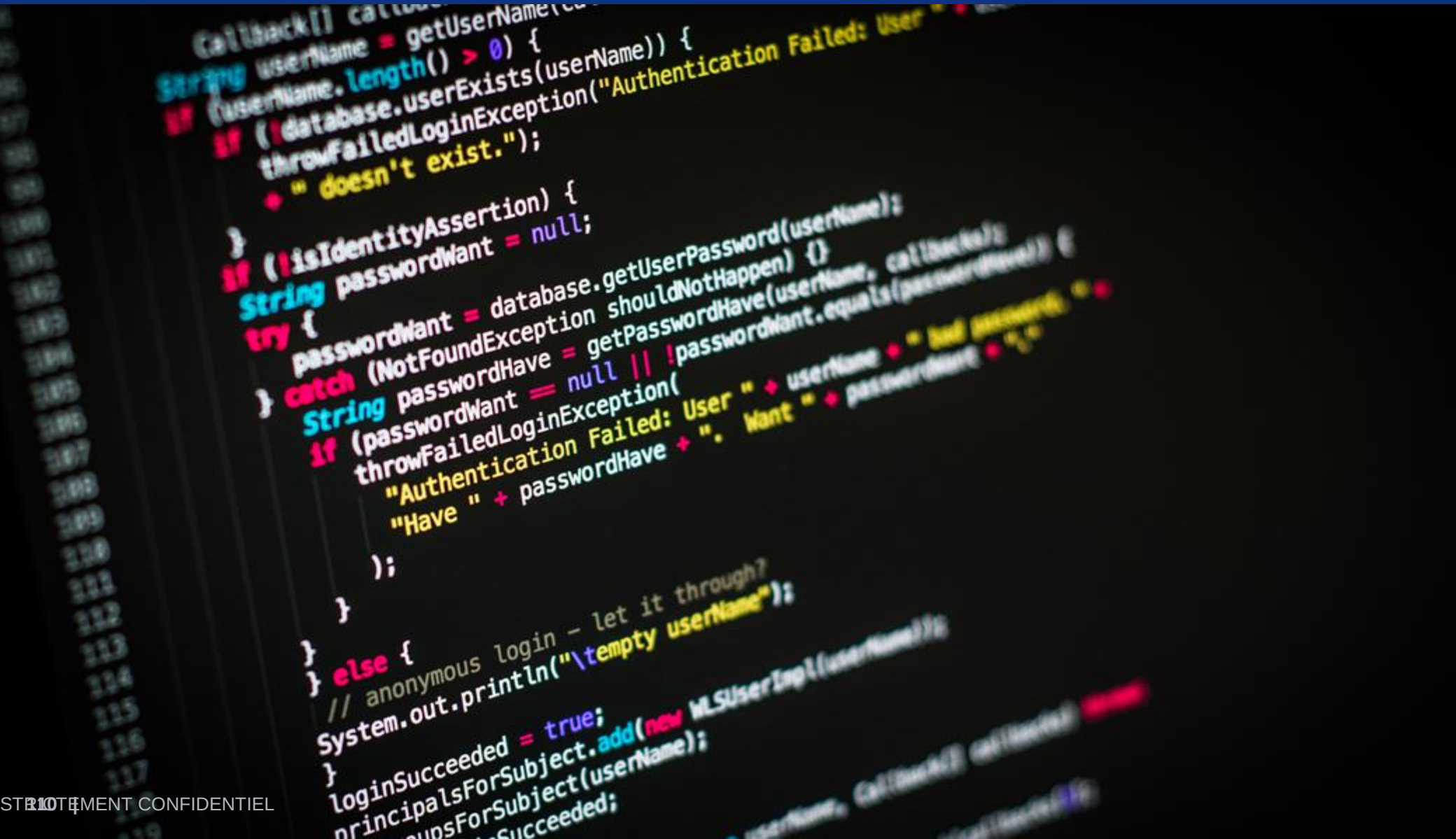
## Avantage:

- Moins de code dupliqué vu qu'on centralise le comportement à un seul endroit.
- Meilleure organisation du code. (Les responsabilités des objets sont mises au bon endroit)

# Message Chains

réinventons / notre métier

## Démo



# Message Chains



## Code Smell

```
def getTotalPrice():
    invoiceTotal = 0

    for item in invoiceItems:
        invoiceTotal += item.getSubTotal()

    if not customer.Adress.Country.isInEurope():
        invoiceTotal += SHIPPING_COST_OUTSIDE_EU

    return  invoiceTotal
```

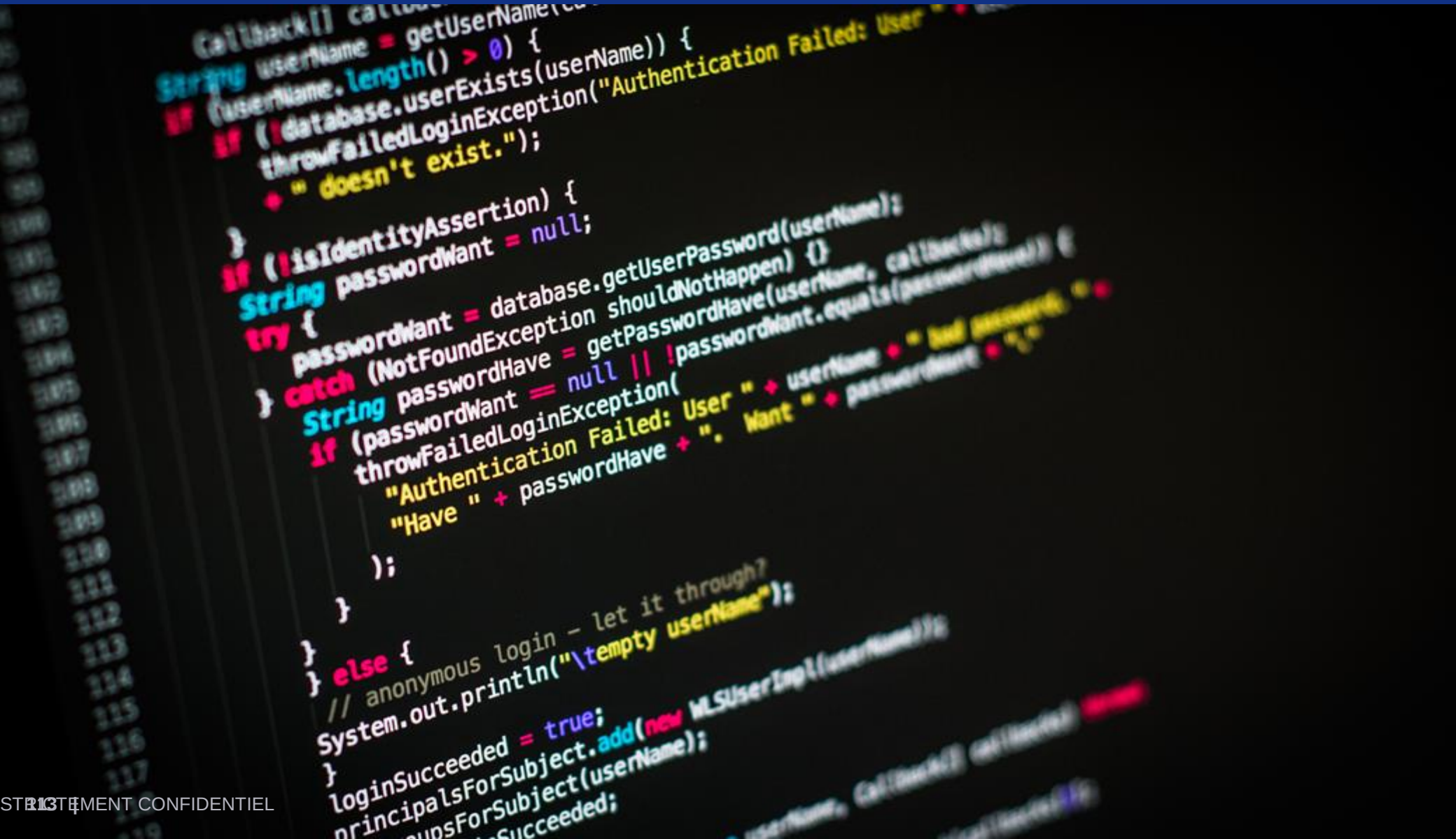
# Message Chains

- **Explication:**
  - Dans le code, vous voyez une série d'appels ressemblant à a.b().c().d()
- **Raison du problème:**
  - Une chaine de messages se produit lorsqu'un client demande un autre objet, cet objet en demande un autre, etc. Ces chaines signifient que le client dépend de la navigation le long de la structure de la classe. Toute modification de ces relations nécessite la modification du client.
- **Avantage:**
  - Réduit les dépendances entre les classes d'une chaine
  - Réduit la quantité de code

# Middle Man

réinventons / notre métier

## Démo



## Code Smell

```
class Customer:
    def __init__(self):
        self.addresses = Addresses()

    def sendMail(self, mailContent, mailType):
        return "To:" + "Content:" + mailContent

    def getAddress(self, mailType):
        if mailType == 1:
            return self.addresses.addressBilling



class Addresses:
    def __init__(self):
        self.addressBilling = Address(21, "Jump street", "Hollywood" )



class Address:
    def __init__(self, number, street, city):
        self.number = None
        self.street = None
        self.city = None
```

# Middle Man

## Explication:

- En voulant bien découper, on peut créer de nombreuses classes. Cependant, l'excès de classe peut devenir contreproductif.

## Raison du problème:

- L'excès de classe intermédiaire complexifie la compréhension.

## Avantage:

- Facilite la compréhension en limitant le nombre de classe à explorer.

Conseil de vétéran : Certains découpage ont une utilité pour l'architecture (Proxy, Adapteur, Découpage technique, …).

# Les Tips à retenir

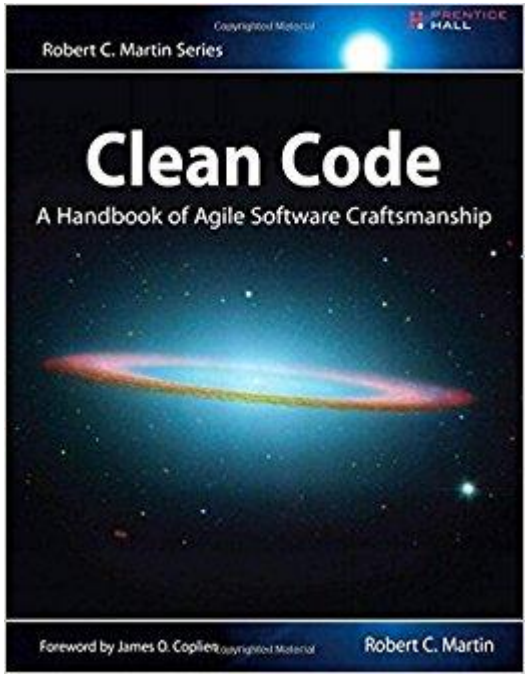


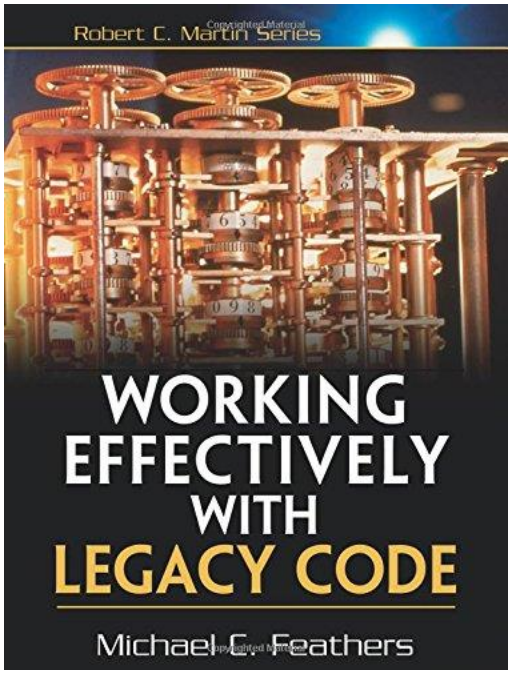
- Faites confiance aux outils de votre IDE (Resharper par exemple), solution d'amélioration automatique, complexité cyclomatique
- Les raccourcis claviers
- Ne vous lancer pas dans une refacto en fin de release
- Ne vous lancer pas dans une refacto sans être couvert par des tests unitaires
- Un renamme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

# References



- 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**



# *Exercice*

- TP n°1

## Définir et importer des packages

```
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

# CLEAN ARCHITECTURE

- Indépendance des frameworks
- Testabilité
- Indépendance UI et Base

**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.

## Convention de nommage:

- modules : nom court, tout en minuscules, tiret du bas si nécessaire. **great_module**
- paquets : nom court, tout en minuscules, tirets du bas déconseillés. **paquet**
- classes : lettres majuscules en début de mot. **MyGreatClass**
- exceptions : similaire aux classes mais avec un Error à la fin. **MyGreatError**
- fonctions : minuscules et tiret du bas : **my_function()**
- méthodes : minuscules, tiret du bas et self en premier paramètre : **my_method(self)**
- arguments des méthodes et fonctions : identique aux fonctions. **my_function(param=False)**
- variables : identique aux fonctions.
- constantes : tout en majuscules avec des tirets si nécessaire. **I_WILL_NEVER_CHANGE**
- privé : précédé de deux tirets du bas : **__i_am_private**
- protégé : précédé d'un tiret du bas : **_i_am_protected**