



# INSTITUT SUPÉRIEUR DE TECHNOLOGIE D'ANTANANARIVO

--oOo--



*Premier Établissement Public d'Enseignement Supérieur  
et de Recherche Habilité et Accrédité à Madagascar par le  
MESUPRES*

## ÉCOLE DU GÉNIE INDUSTRIEL

Mention : **GÉNIE DES SYSTÈMES INDUSTRIELS**

## RAPPORT DU PROJET LOGICIEL DE CALCUL TECHNIQUE :

## ÉSTIMATEUR DE CONSOMMATION D'EAU

Membres du groupe N°IV :

ANDRIANAMBININA Liantsoa	GSI-MP	N°02
RAKOTOMALALA Andry Ambinintsoa	GSI-ER	N°09
RAKOTOMAVO Finaritra Vola	GSI-ER	N°10

## **TABLE DES MATIERES**

Table des matières .....	i
Introduction.....	1
<b><u>PARTIE I- CADRAGE DU PROJET.....</u></b>	<b><u>2</u></b>
<b>I-I- PRESENTATION DU SUJET.....</b>	<b>2</b>
I-I-1- ÉNONCE DU PROBLEME .....	2
I-I-2- OBJECTIFS DU PROJET .....	2
I-I-3- PORTEE DU PROJET .....	2
I-I-4- ATTENTES ET LIVRABLES.....	2
<b>I-II- PRESENTATION DU LANGAGE DE PROGRAMMATION PYTHON .....</b>	<b>2</b>
I-II-1- QUELQUES CARACTERISTIQUES CLES DE PYTHON .....	2
<b>I-III- PRESENTATION DU LOGICIEL.....</b>	<b>3</b>
I-III-1- INTRODUCTION A VISUAL STUDIO CODE .....	3
I-III-2- CARACTERISTIQUES ET FONCTIONNALITES .....	3
I-III-3- AVANTAGES DE L'UTILISATION DE VS CODE POUR NOTRE PROJET .....	4
<b><u>PARTIE II- ÉLABORATION DU PROGRAMME .....</u></b>	<b><u>5</u></b>
<b>II-I- FONCTIONNEMENT DU CODE .....</b>	<b>5</b>
II-I-1- INTRODUCTION.....	5
II-I-2- INTERPRETATION DES COMMANDES .....	5
<b>II-II- STRUCTURE DU CODE .....</b>	<b>10</b>
II-II-1- FONCTION UTILITAIRE « IS_FLOAT ».....	10
II-II-2- CLASSE « SHELL ».....	10
II-II-3- METHODE D'AIDE .....	10
II-II-4- MECANISME DE STOCKAGE .....	10
II-II-5- MODELES DE DONNEES .....	12
II-II-6- METHODES DE COMMANDES.....	15
<b>II-III- EXECUTION DU PROGRAMME.....</b>	<b>16</b>
Conclusion .....	20

## **INTRODUCTION**

L'eau, une ressource vitale et irremplaçable, demeure au cœur des préoccupations contemporaines liées à la durabilité environnementale et à la gestion des ressources naturelles. Face à l'accroissement de la pression démographique et aux défis posés par le changement climatique, il devient impératif de mieux appréhender et gérer notre consommation d'eau, tant à l'échelle domestique qu'industrielle.

Dans ce contexte, le développement d'outils informatiques permettant d'estimer avec précision la consommation d'eau revêt une importance cruciale. Cela permet non seulement d'évaluer et de comprendre nos besoins en eau, mais aussi de mettre en place des mesures efficaces pour une utilisation plus responsable de cette ressource.

Le présent rapport se concentre sur l'analyse et la présentation d'un tel outil, développé en utilisant le langage de programmation Python. Cet outil vise à fournir une estimation précise de la consommation d'eau, que ce soit dans un cadre domestique ou industriel. En combinant des algorithmes sophistiqués avec des données pertinentes, cet outil offre une approche pratique et informatisée pour évaluer la consommation d'eau.

Au fil des sections suivantes, nous examinerons en détail les fonctionnalités de cet outil ainsi que son architecture. Ensuite, nous aborderons les perspectives d'amélioration envisagées pour cet outil spécifique. En regroupant ces informations, ce rapport vise à offrir un aperçu complet de l'outil d'estimation de la consommation d'eau développé en Python.

## **PARTIE I- CADRAGE DU PROJET**

Dans cette première partie du rapport, nous allons mettre en évidence les caractéristiques du projet, c'est-à-dire l'énoncé du sujet, ces objectifs ainsi que le logiciel que nous avons utilisé pour l'élaboration du projet.

### **I-I- PRESENTATION DU SUJET**

#### **I-I-1- Énoncé du problème**

Dans le cadre de notre projet, nous avons été chargés de développer un outil informatique permettant d'estimer la consommation d'eau, aussi bien pour un usage domestique que pour les applications industrielles.

#### **I-I-2- Objectifs du projet**

Notre objectif principal est de concevoir un outil convivial et précis, capable de fournir des estimations fiables de la consommation d'eau dans différents contextes. Nous visons à fournir un moyen efficace de comprendre et de gérer cela.

#### **I-I-3- Portée du projet**

Nous concentrerons notre attention sur le développement d'un outil codé en Python, offrant des fonctionnalités d'estimation de la consommation d'eau basées sur des algorithmes et des données pertinentes. Bien que notre outil puisse être utilisé à diverses échelles, nous mettrons particulièrement l'accent sur son utilisation dans des foyers et les entreprises.

#### **I-I-4- Attentes et livrables**

Nous prévoyons de livrer un outil fonctionnel qui pourra être utilisé par les utilisateurs pour estimer leur consommation d'eau avec précision. En plus de l'outil, nous fournirons une documentation détaillée pour les utilisateurs dans son utilisation, sa compréhension et son déploiement.

### **I-II- PRESENTATION DU LANGAGE DE PROGRAMMATION PYTHON**

Python est un langage de programmation de haut niveau, interprété et orienté objet, créé par Guido van Rossum et publié pour la première fois en 1991. Connu pour sa syntaxe claire et intuitive, Python est largement utilisé dans divers domaines, notamment le développement web, la science des données, l'intelligence artificielle, l'automatisation des tâches et bien plus.

#### **I-II-1- Quelques caractéristiques clés de Python**

##### **a) Facilité d'apprentissage**

Il est réputé pour sa simplicité et sa lisibilité du code, ce qui en fait un excellent choix pour ceux qui débutent en programmation.

##### **b) Polyvalence**

Python offre une vaste gamme de bibliothèques et de frameworks qui le rendent adapté à de nombreuses applications, des scripts simples aux projets complexes.

### c) Vaste écosystème

Python bénéficie d'une communauté active et dynamique qui contribue constamment à son écosystème. Il existe de nombreuses ressources, forums, tutoriels et packages disponibles pour les utilisateurs de Python.

### d) Interprétation

Il est un langage interprété, ce qui signifie que le code source est traduit en langage machine à la volée par un interpréteur. Cela rend Python portable et facilite le développement interactif.

### e) Orienté objet

Python prend en charge la programmation orientée objet, ce qui permet une modélisation efficace des problèmes complexes en utilisant des concepts tels que les classes et les objets.

## **I-III- PRESENTATION DU LOGICIEL**

Dans cette section, nous allons discuter du logiciel à savoir le Visual Studio Code que nous avons utilisé pour élaborer notre outil d'estimateur de consommation d'eau qui est codé en Python.

### **I-III-1- Introduction à Visual Studio Code**

Visual Studio Code ou encore VS Code est un environnement de développement intégré développé par Microsoft. Il s'agit d'un outil polyvalent, largement utilisé par les développeurs pour créer une variété d'applications logicielles, y compris des projets Python comme le nôtre.

### **I-III-2- Caractéristiques et fonctionnalités**

#### a) Interface utilisateur intuitive

VS Code offre une interface utilisateur moderne et intuitive, ce qui en fait un outil convivial même pour les débutants.

#### b) Personnalisable

Il hautement personnalisable grâce à une vaste gamme d'extensions disponibles, permettant aux développeurs de configurer l'interface selon leurs préférences et leurs besoins spécifiques.

#### c) Prise en charge du langage Python

VS Code offre une prise en charge native du langage Python, ce qui en fait un choix idéal pour le développement de projets Python comme le nôtre. Il propose des fonctionnalités avancées telles que la coloration syntaxique, l'auto-complétions, le débogage, etc.

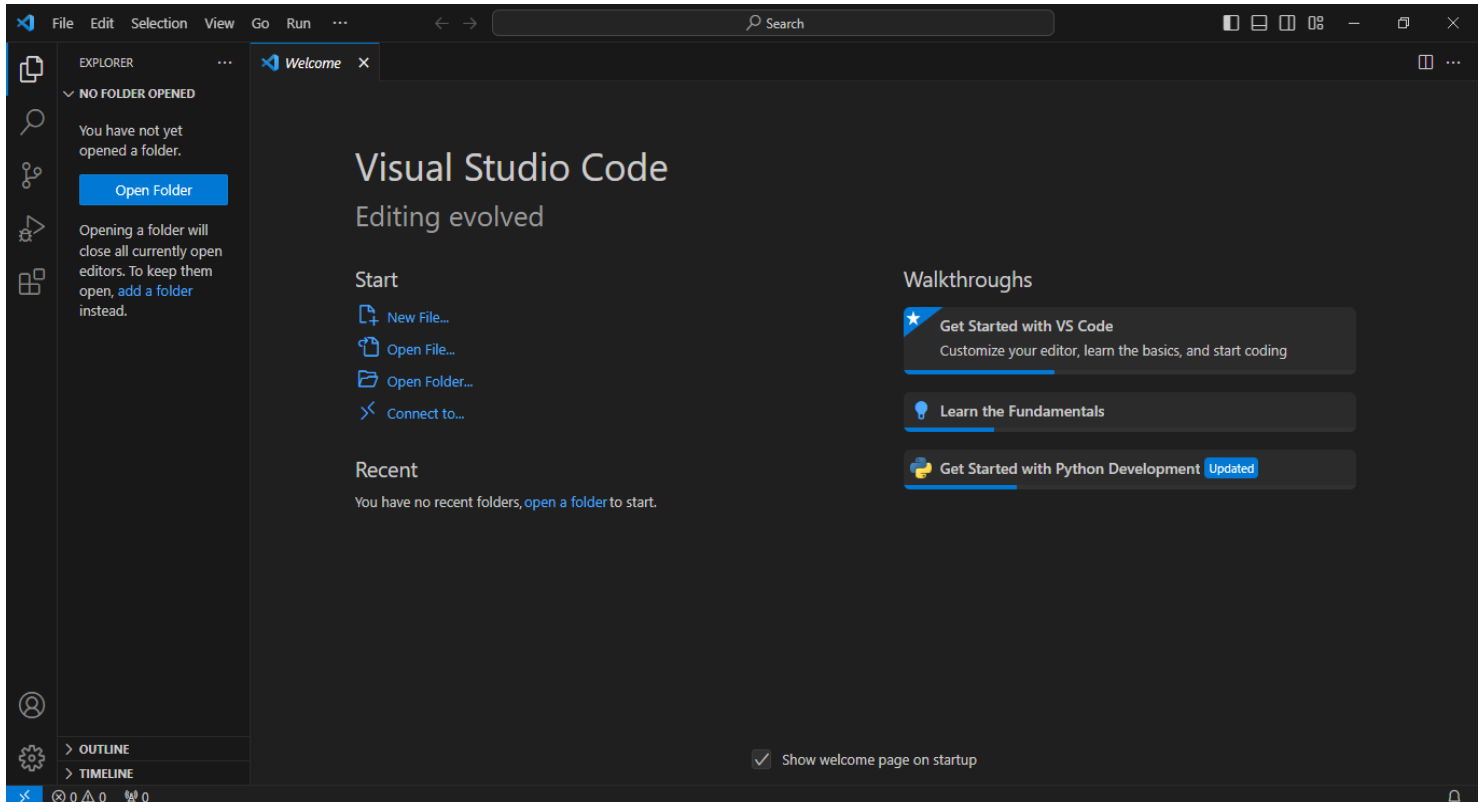
### **I-III-3- Avantages de l'utilisation de VS Code pour notre projet**

#### **a) Productivité améliorée**

L'interface conviviale et les fonctionnalités avancées de VS Code ont contribué à améliorer notre productivité tout au long du processus de développement de notre outil d'estimation de la consommation d'eau.

#### **b) Développement efficace**

La prise en charge native du langage Python dans VS Code, associée à ses fonctionnalités de débogage et d'auto-complétions, nous a permis de développer, tester et déployer notre outil de manière efficace et sans heurts.



En résumé, l'utilisation de Visual Studio Code comme environnement de développement pour ce projet s'est avérée être un choix judicieux car ses fonctionnalités avancées, sa convivialité et sa flexibilité ont grandement contribué à la réussite de notre projet d'estimation de la consommation d'eau en Python.

## PARTIE II- ÉLABORATION DU PROGRAMME

Maintenant que nous avons posé les bases de notre projet et présenté l'environnement de développement que nous avons utilisé, Visual Studio Code, il est temps de plonger dans l'élaboration de notre programme d'estimation de la consommation d'eau. Dans cette partie, nous explorerons le processus de développement, les choix architecturaux, les algorithmes utilisés, ainsi que les défis rencontrés et les solutions apportées. En examinant de près le développement de notre programme. Nous découvrirons comment chaque étape, chaque fonction ont contribué à la création de notre outil pour estimer la consommation d'eau. Sans plus tarder, plongeons-nous dans les détails de la conception et l'implémentation de notre programme en Python.

### II-I- FONCTIONNEMENT DU CODE

#### II-I-1- Introduction

Le programme est un outil de commande développé en Python pour estimer la consommation d'eau. Il utilise une architecture basée sur des commandes où l'utilisateur peut saisir des commandes spécifiques pour créer, afficher, mettre à jour ou détruire des objets représentant des appareils et des salles.

#### II-I-2- Interprétation des commandes

Lorsque l'utilisateur lance le programme, il est accueilli par une invite de commande « (estimateur) ». Il peut ensuite saisir différentes commandes telles que « create », « show », « update », « destroy », « all » et « precise », suivies des arguments appropriés pour effectuer des opérations spécifiques sur l'objet.

##### a) Création d'objets

Utilisation de la commande « create » pour créer des objets de différentes classes, notamment « BaseModel », « Appareil » et « Salle ». Cela permet aux utilisateurs d'ajouter de nouveaux appareils ou salle à leur système de gestion d'eau.

```
41  def do_create(self, arg):
42      """
43      Crée un objet de son choix
44      """
45      arg_list = arg.split()
46      if arg == "":
47          print("** Manque du nom de la classe **")
48      elif arg not in self.class_list:
49          print("** Cette classe n'existe pas **")
50      else:
51          obj = eval(arg)()
52          obj.save()
53          print(obj)
54
55  def help_create(self):
56      """Print la documentation de create"""
57      print("create <arg_1> : Crée un objet de type arg_1")
58
```

### b) Affichage d'objets

La commande « show » permet d'afficher les détails d'un objet spécifique en fournissant son nom de classe et son identifiant unique. Cela permet aux utilisateurs de visualiser les informations sur un appareil ou une salle particulière dans leur système.

```
59 def do_show(self, arg):
60     """
61     Print la representation de l'objet basée sur le nom de la classe et son identifiant unique
62     """
63     arg_list = arg.split()
64     if not arg_list:
65         print("*** Manque de l'identifiant ***")
66         return
67
68     obj_id = arg_list[0]
69     found_obj = None
70     for obj in storage.all().values():
71         if obj.id == obj_id:
72             found_obj = obj
73             break
74
75     if found_obj is None:
76         print("Objet avec l'ID '{}' n'a pas été trouvé.".format(obj_id))
77     else:
78         print(found_obj)
79
80 def help_show(self):
81     """Print la documentation de show"""
82     print("show <id_1>: Montre la représentation de l'objet ayant pour identifiant id_1")
83
```

### c) Suppression d'objets

Avec la commande « destroy », les utilisateurs peuvent supprimer un objet spécifique en fournissant son nom de classe et son identifiant unique. Cela offre une fonctionnalité de gestion des données, permettant aux utilisateurs de supprimer des appareils ou des salles de leur système en cas de besoins.

```
84 def do_destroy(self, arg):
85     """
86     Supprime un objet selon son nom de classe et son identifiant unique
87     (sauvegarde les changements effectués)
88     """
89     arg_list = arg.split()
90     if arg == "":
91         print("*** Manque du nom de la classe ***")
92     elif arg_list[0] not in self.class_list:
93         print("*** Cette classe n'existe pas ***")
94     elif len(arg_list) < 2:
95         print("*** Manque de l'identifiant unique ***")
96     else:
97         obj_key = arg_list[0] + "." + arg_list[1]
98         if obj_key in storage.all():
99             storage.all().pop(obj_key)
100             storage.save()
101             print("Detruit")
102             return
103         print("*** L'objet n'existe pas ***")
104
105 def help_destroy(self):
106     """Print la documentation de destroy"""
107     print("destroy <arg_1> <id_1>: Supprime un objet selon son nom de classe arg_1 et son identifiant unique id_1")
108
```



#### d) Liste d'objets

La commande « all » permet de lister tous les objets ou les objets d'une classe spécifique. Cela permet aux utilisateurs de visualiser l'ensemble des appareils ou salles enregistrés dans leur système, facilitant ainsi la gestion globale de la consommation d'eau.

```
109 def do_all(self, arg):
110     """
111     Prints toutes les representations de tous les objets selon ou non leur nom de classe
112     """
113     if arg == "":
114         for obj in storage.all().values():
115             print(str(obj))
116     elif arg in self.class_list:
117         for obj in storage.all().values():
118             if obj.__class__.__name__ == arg:
119                 print(str(obj))
120     else:
121         print("*** L'objet n'existe pas ***")
122
123 def help_all(self):
124     """Print la documentation de all"""
125     print("all <arg_1>: Prints toutes les representations de tous les objets selon ou non leur nom de classe arg_1. Si arg_1 n'est pas donné")
126
```

#### e) Mise à jour d'objets

Avec la commande « update », les utilisateurs peuvent mettre à jour les attributs d'un objet spécifique en fournissant son nom de classe, son identifiant unique, le nom de l'attribut à mettre à jour et sa nouvelle valeur. Cela offre une flexibilité pour ajuster les informations sur les appareils ou les salles selon les besoins.

```
127 def do_update(self, arg):
128     """
129     Met à jour l'objet selon son nom de classe et de son identifiant en ajoutant ou mettant à jour son attribut
130     (Enregistre les changements effectués)
131     """
132     arg_list = arg.split(' ')
133     if arg == "":
134         print("*** Manque du nom de la classe ***")
135     elif arg_list[0] not in self.class_list:
136         print("*** Cette classe n'existe pas ***")
137     elif len(arg_list) < 2:
138         print("*** Manque de l'identifiant unique ***")
139     elif len(arg_list) < 3:
140         print("*** Manque du nom de l'attribut ***")
141     elif len(arg_list) < 4:
142         print("*** Manque de la valeur de l'attribut ***")
143     else:
144         obj_key = arg_list[0] + "." + arg_list[1]
145         if obj_key in storage.all():
146             obj = storage.all()[obj_key]
147             attribute_name = arg_list[2]
148             new_value = arg_list[3][1:-1]
149             if attribute_name in ['cons', 'user']:
150                 if not new_value.isdigit() and not is_float(new_value):
151                     print("La valeur de l'attribut '{}' doit être un entier ou un nombre décimal.".format(attribute_name))
152                     return
153             setattr(obj, attribute_name, new_value)
154             storage.save()
155             print("Mise à jour de l'attribut '{}' de l'objet {} avec la valeur {}".format(attribute_name, obj_key, new_value))
156         else:
157             print("*** L'objet n'existe pas ***")
158
```

#### f) Estimation de la consommation

La méthode « do\_estimate » permet de fournir une estimation de la consommation d'eau totale dans le système ou pour un type spécifique d'objets. Voici comment elle fonctionne :

→ L'utilisateur entre la commande « estimate » dans l'interface.

→ Si aucun argument n'est fourni, la méthode calcule la consommation totale en parcourant tous les objets présents dans le système et en additionnant leur consommation respective.

→ Puis si un argument est fourni et qu'il correspond à une classe valide dans la liste des classes disponibles, la méthode calcule la consommation totale pour cette classe spécifique d'objets en additionnant leur consommation respective.

Le calcul est effectué à partir de cette formule :

$$T = \sum_{i=1}^n C_i * U_i$$

Tels que :

- **$T$** : la consommation d'eau totale pour les objets de classe précisée par l'utilisateur (Appareil ou Salle), tous les objets existants si aucune classe n'est précisée.
- **$C_i$** : la consommation en eau moyenne par personne de l'objet  $i$ .
- **$U_i$** : le nombre d'utilisateur utilisant l'objet  $i$ .
- **$i$** : allant de 1 à  $n$ , signifiant le  $i$ -ème objet, de classe Appareil et/ou Salle, dans notre sauvegarde.
- **$n$** : le nombre d'objets de type Appareil et/ou Salle.

```
164 def do_estimate(self, arg):
165     """
166     Fait une estimation de la consommation moyenne des utilisateurs
167     """
168     estim = 0.0
169     if arg == "":
170         for obj in storage.all().values():
171             estim += float(obj.cons) * float(obj.user)
172         print("La consommation totale de tout le systeme est : {:.2f} L".format(estim))
173     elif arg in self.class_list:
174         for obj in storage.all().values():
175             if obj.__class__.__name__ == arg:
176                 estim += float(obj.cons) * float(obj.user)
177         print("La consommation totale de tous les '{}' est : {:.2f} L".format(arg, estim))
178     else:
179         print("*** L'objet n'existe pas ***")
180
181 def help_estimate(self):
182     """Print la documentation de estimate"""
183     print("estimate <arg_1>: Print une estimation de la consommation en eau de arg_1 des utilisateurs. Si arg_1 n'est pas donné, print la con")
184
```

### g) Estimation précise de la consommation

La commande « precise » permet à l'utilisateur de spécifier les identifiants uniques des appareils ou des salles pour lesquels il souhaite estimer la consommation d'eau totale. Le programme calcule ensuite la consommation d'eau totale pour les objets précisés et affiche le résultat.

Le calcul se fait par cette formule :

$$T = \sum_1^n C_i * U_i$$

Tels que :

- **$T$**  : la consommation d'eau totale pour les objets précisés (appareils et/ou salles)
- **$C_i$**  : la consommation en eau moyenne par personne de l'objet (appareil ou salle) ayant comme identifiant l'i-ème argument passé dans la commande « precise » (exemple : « precise identifiant\_1 identifiant\_2 »). Ici  $C_1$  correspond à la consommation en eau moyenne par personne de l'objet ayant comme identifiant identifiant\_1).
- **$U_i$**  : le nombre d'utilisateur utilisant l'objet (appareil ou salle) ayant comme identifiant l'i-ème argument passé dans la fonction « precise ».
- **$n$**  : le nombre d'identifiant passé dans la commande « precise »

```
185 def do_precise(self, arg):
186     """
187     Fait un calcul de la consommation des appareils et/ou salles précisés
188     """
189     arg_list = arg.split()
190     if not arg_list:
191         print("*** Manque de l'identifiant ***")
192         return
193
194     total_conso = 0.0
195     for obj_id in arg_list:
196         found_obj = None
197         for obj in storage.all().values():
198             if obj.id == obj_id:
199                 found_obj = obj
200                 break
201
202         if found_obj is None:
203             print("Objet avec l'ID '{}' n'a pas été trouvé.".format(obj_id))
204             return
205         else:
206             conso = float(found_obj.conso) * float(found_obj.user)
207             total_conso += conso
208     print("La consommation totale des objets précisés est : {:.2f} L".format(total_conso))
209
210 def help_precise(self):
211     """Print la documentation de precise"""
212     print("precise <id_1> <id_2> <id_3> ...: Print une estimation de la consommation en eau des objets ayant pour id : id_1, id_2, ... (effe
213
```

En résumé, notre programme fournit une interface pratique et conviviale pour créer, afficher, supprimer, sauvegarder, lister et mettre à jour les objets à notre projet d'estimation de la consommation d'eau. Cela permet aux utilisateurs de gérer efficacement leur système de gestion de l'eau et d'obtenir des estimations précises de leur consommation d'eau domestique ou industrielle.

## II-II- STRUCTURE DU CODE

Le code est structuré de manière à séparer clairement les fonctionnalités de l'interface en ligne de commande et la manipulation des objets et des données. Nous allons voir quelques aperçus de la structure de notre code.

### II-II-1- Fonction utilitaire « is float »

Cette fonction vérifie si une valeur peut être convertie en nombre flottant. Elle sera utilisée pour valider les entrées utilisateurs lors de la mise à jour des attributs.

```
7 def is_float(value):
8     """Vérifie si value peut être convertie en float"""
9     try:
10         float(value)
11         return True
12     except ValueError:
13         return False
```

### II-II-2- Classe « Shell »

Située dans console.py, cette classe hérite de « cmd.Cmd » et définit les différentes commandes disponibles dans notre interface en ligne de commande. Chaque commande est implémentée comme une méthode de cette classe, avec une documentation détaillée expliquant son fonctionnement.

### II-II-3- Méthode d'aide

Pour cette méthode, nous avons utilisé les méthodes « help » qui vont afficher l'aide et la documentation des différentes commandes. Prenons un exemple dans le code :

```
25 def help_print(self):
26     """Print la documentation de print"""
27     print("print <arg_1> : print arg_1")
28
```

### II-II-4- Mécanisme de stockage

Notre code utilise un mécanisme de stockage pour conserver les objets créés par l'utilisateur entre les différentes sessions d'utilisation du programme. La classe « FileStorage », dans models/engine/file\_storage.py, fournit des méthodes pour sauvegarder, charger et manipuler les données stockées.

Dans le code nous avons défini une classe « FileStorage » qui va gérer le stockage et le changement des données de l'outil dans un fichier JSON. Voici une explication de chaque partie du code :

#### a) Attributs des classes

- « \_\_file\_path » : un chemin du fichier de sauvegarde JSON.
- « \_\_objects » : dictionnaire contenant tous les objets de l'outil, avec les clés sous la forme « <nom de classe>.id ».

```

12     __file_path = "sauvegarde.json"
13     __objects = {}
14

```

## b) Méthodes

- « all(self) » : retourne le dictionnaire contenant tous les objets.

```

15  def all(self):
16      """Retourne le dictionnaire de tous les objets"""
17      return FileStorage.__objects
18

```

- « new(self, obj) » : ajoute un nouvel objet au dictionnaire « \_\_objects ».

```

19  def new(self, obj):
20      """Met dans __objects l'objet avec comme cle : <nom de la classe de l'objet>.id"""
21      key = obj.__class__.__name__ + "." + obj.id
22      FileStorage.__objects[key] = obj
23

```

- « save(self) » : sauvegarde tous les objets présents dans « \_\_objects » dans le fichier JSON.

```

24  def save(self):
25      """
26      Sauvegarde tous les objets en fichier JSON
27      """
28      obj_dict = {}
29      for key, value in FileStorage.__objects.items():
30          obj_dict[key] = value.to_dict()
31
32      with open(FileStorage.__file_path, "w", encoding="utf-8") as f:
33          json.dump(obj_dict, f)
34

```

- « reload(self) » : charge les objets à partir du fichier JSON dans « \_\_objects ». Ces méthodes permettent de gérer la sauvegarde et le chargement des objets de l'outil, offrant ainsi une méthode de persistance des données entre les différentes son exécution.

```

35  def reload(self):
36      """
37      Transforme le fichier JSON en __objects (Seulement si ce fichier existe)
38      """
39      classes = {'BaseModel': BaseModel, 'Appareil': Appareil, 'Salle': Salle}
40      try:
41          with open(FileStorage.__file_path, "r") as f:
42              object_dict = json.load(f)
43              for key, val in object_dict.items():
44                  self.__objects[key] = classes[val['__class__']](**val)
45      except FileNotFoundError:
46          pass

```

## c) Utilisation des classes des modèles

Les classes des modèles (« BaseModel », « Appareil », « Salle ») sont utilisées pour instancier les objets lors du chargement à partir du fichier JSON.

## II-II-5- Modèles de données

Notre code importe les classes « BaseModel », « Appareil » et « Salle » pour présenter les différents types d'objets manipulés dans notre système de gestion de l'eau. Chaque classe définit les attributs et les méthodes nécessaires pour créer, afficher, mettre à jour et supprimer les objets correspondants.

```
1  import cmd
2  from models.base_model import BaseModel
3  from models.appareil import Appareil
4  from models.salle import Salle
5  from models import storage
6
```

### a) BaseModel

Ce module importe les modules « models » qui est le module locale contenant d'autres parties de l'outil, et « uuid » qui est un module de la bibliothèque standard Python utilisé pour générer des identifiants uniques.

- **Attribut « id »** : Cet attribut est défini mais pas initialisé. Il représente l'identifiant unique de chaque objet de la classe. Il obtient une valeur unique générée par la méthode `uuid.uuid4()` lors de la création d'un nouvel objet.

```
def __init__(self, *arg, **kwargs):
    """
    self : l'objet en question
    arg : non utilisé
    kwarg : dictionnaire de valeur de nos attributs (comme nom et consommation)
    """
    if kwargs and len(kwargs) > 0:
        for key, value in kwargs.items():
            self.__dict__[key] = value
    else:
        self.id = str(uuid.uuid4())
        models.storage.new(self)
```

- **Méthode « \_\_init\_\_ »** : Le constructeur initial de la classe initialise les attributs de l'objet. Si les arguments nommés sont passés (« kwargs »), les attributs sont initialisés avec les valeurs spécifiées. Sinon, un identifiant unique est généré pour l'objet et il est enregistré dans le système de stockage (« models.storage »).

```
17  def __init__(self, *arg, **kwargs):
18      """
19      self : l'objet en question
20      arg : non utilisé
21      kwarg : dictionnaire de valeur de nos attributs (comme nom et consommation)
22      """
23      if kwargs and len(kwargs) > 0:
24          for key, value in kwargs.items():
25              self.__dict__[key] = value
26      else:
27          self.id = str(uuid.uuid4())
28          models.storage.new(self)
```

- **Méthode « `__str__` »** : Cette méthode définit la représentation sous forme de chaîne de caractère de l'objet. Elle retourne une chaîne formatée avec le nom de la classe, le nom de l'objet, son identifiant, le nombre d'utilisateurs et sa consommation.

```
def __str__(self):
    """
    Retourne la representation de comment BaseModel devrait etre print-er
    """
    return ("L'objet de type '{}' de nom '{}' d'identifiant: {} est utilise par {} personne(s). Consommation: {} L/personne ".format(self.__class__.__name__, self.nom, self.id, self.user, self.cons))
```

- **Méthode « `save` »** : Cette méthode appelle la méthode « `save()` » du module « `models.storage` », qui sauvegarde l'objet dans le système de stockage .

```
36 def save(self):
37     """
38     Sauvegarde l'objet
39     """
40     models.storage.save()
```

- **Méthode « `to_dict` »** : Cette méthode convertit l'objet en un dictionnaire contenant tous ses attributs, ainsi que le nom de la classe.

```
42 def to_dict(self):
43     """
44     Retourne un dictionnaire contenant tous les clés/valeurs de __dict__ de l'objet
45     """
46     base_dict = self.__dict__.copy()
47     base_dict["__class__"] = self.__class__.__name__
48     return base_dict
```

Ce code définit une structure de base pour la création d'autres classes dans le projet. Il permet de générer des objets avec des identifiants uniques, de les sauvegarder et de les représenter sous forme de chaînes de caractère. Il suit le modèle de conception de classe de base pour faciliter l'extension et la personnalisation des fonctionnalités dans d'autres parties de l'outil.

#### b) Salle

Ce code définit une classe appelée « `Salle` », qui hérite de la classe « `BaseModel` ».

- **Importation de « `BaseModel` »** : Le module importe la classe « `BaseModel` » du module « `model.base_model` ». Cela signifie que la classe « `Salle` » va hériter de toutes les fonctionnalités définies dans « `BaseModel` ».

```
4 from models.base_model import BaseModel
```

- **Classe « `Salle` »**

✓ Attributs :

« nom » : Le nom de la salle. Par défaut, il est défini comme « `sans nom` ».

« user » : Le nombre d'utilisateur de la salle. Par défaut, il est initialisé à 0.

« cons » : La consommation en eau de la salle par utilisateur. Par défaut, il est initialisé à 0.

✓ Méthode « \_\_init\_\_ » : Le constructeur de la classe appelle d'abord le constructeur de la classe parent « BaseModel » en utilisant « `super().__init__(**kwargs)` ». Cela permet d'initialiser les attributs de base de la classe parent. Ensuite, la classe « Salle » peut éventuellement ajouter des attributs spécifiques ou modifier ceux hérités de la classe parent en fonction des valeurs passées dans « `kwargs` ».

```
6 class Salle(BaseModel):
7     """
8     L'objet Salle
9
10    Attribut(s):
11        nom : nom de la salle
12        user : nombre d'utilisateur utilisant la salle
13        cons : consommation en eau de la salle de chaque utilisateur
14    """
15    def __init__(self, **kwargs):
16        super().__init__(**kwargs)
17
18    nom = "SANS NOM"
19    user = 0
20    cons = 0
```

### c) Appareil

Ce code définit une classe appelée « Appareil », qui hérite également de la classe « BaseModel ».

- **Importation de « BaseModel »** : Le module importe la classe « BaseModel » du module « `model.base_model` ». Cela signifie que la classe « Appareil » va hériter de toutes les fonctionnalités définies dans « BaseModel ».

```
4 from models.base_model import BaseModel
5
```

- **Classe « Appareil »**

✓ Attributs :

« Nom » : Le nom de la salle. Par défaut, il est défini comme « sans nom ».

« User » : Le nombre d'utilisateur de la salle. Par défaut, il est initialisé à 0.

« Cons » : La consommation en eau de l'appareil par utilisateur. Par défaut, il est initialisé à zéro.

✓ Méthode « \_\_init\_\_ » : Le constructeur de la classe appelle d'abord le constructeur de la classe parent « BaseModel » en utilisant « `super().__init__(**kwargs)` ». Cela permet d'initialiser les attributs de base de la classe parent. Ensuite, la classe « Appareil » peut éventuellement ajouter des attributs spécifiques ou modifier ceux hérités de la classe parent en fonction des valeurs passées dans « `kwargs` ».



```

6 class Appareil(BaseModel):
7     """
8     L'objet appareil
9
10    Attribut(s):
11        nom : nom de l'appareil
12        user : nombre d'utilisateur utilisant l'appareil
13        cons : consommation en eau de l'appareil de chaque utilisateur
14    """
15    def __init__(self, **kwargs):
16        super().__init__(**kwargs)
17
18    nom = "SANS NOM"
19    user = 0
20    cons = 0

```

## II-II-6- Méthodes de commandes

Chaque méthode de la classe « shell » correspond à une commande spécifique par exemple « do\_create », « do\_show », « do\_update », ... Chaque méthode effectue une opération spécifique sur les objets en fonction des arguments fournis par l'utilisateur.

En combinant ces éléments, notre code offre une interface conviviale et fonctionnelle pour gérer les objets liés à notre projet d'estimation de la consommation d'eau, tout en assurant la persistance et l'intégrité des données à travers un mécanisme de stockage.

## II-III- EXECUTION DU PROGRAMME

Maintenant que nous avons défini les classes de base ainsi que les classes spécifiques pour les salles et les appareils, nous pouvons passer à l'étape finale : l'exécution du programme. Nous avons créé une structure solide pour notre outil, avec des classes bien définies et des fonctionnalités de base telles que la création, l'affichage et la mise à jour des objets. Dans cette section, nous allons voir comment utiliser ces classes et fonctionnalités pour estimer la consommation d'eau dans différents contextes domestiques ou industriels.

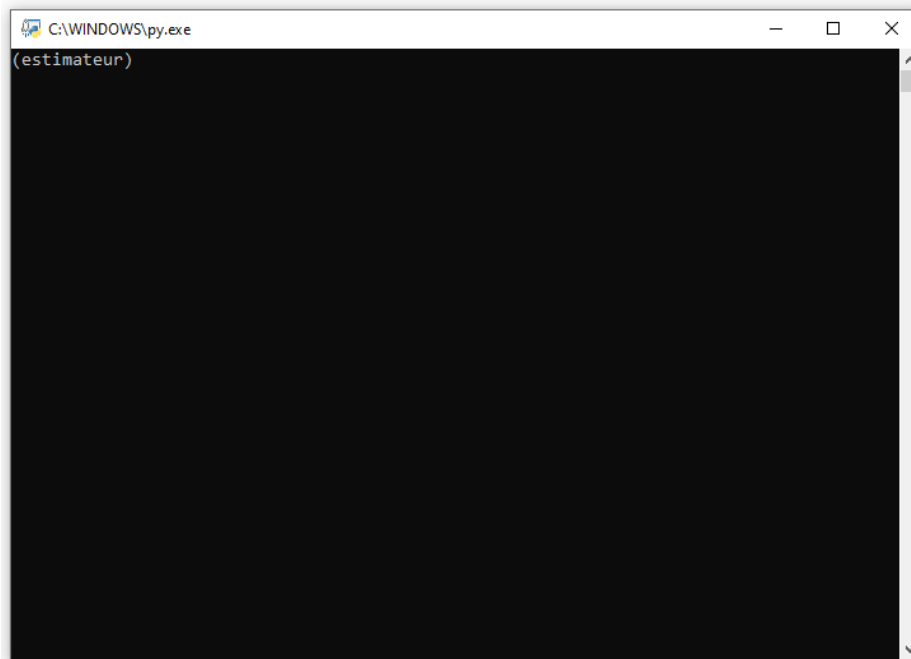
Prenons comme exemple les valeurs suivantes pour l'exécution de notre programme.

Appareils/Salle	Douche	Cuisine	Machine à laver
Consommation moyenne [Litres/personne]	8	5	10
Nombre d'utilisateurs	6		

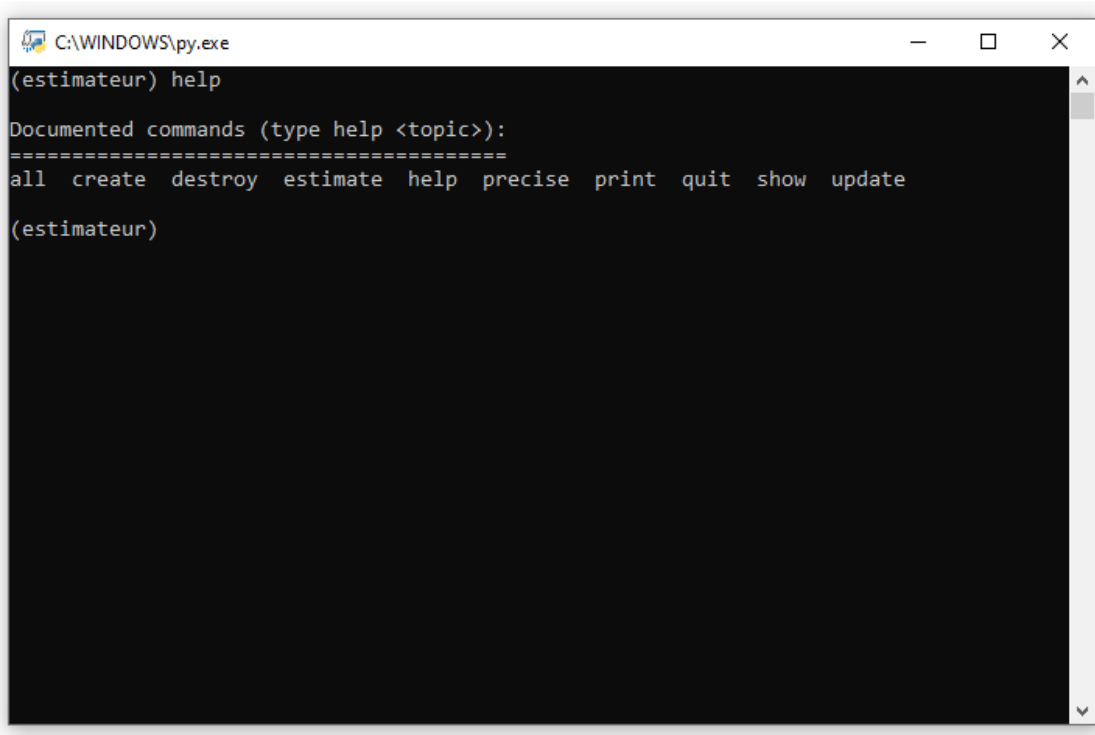
Nous pouvons directement lancer le programme en exécutant le fichier console.py.

models	4/16/2024 3:16 PM	File folder	
.gitignore	4/15/2024 11:54 AM	Text Document	1 KB
console.py	4/16/2024 5:58 PM	Python File	8 KB
README.md	4/16/2024 5:43 PM	MD File	2 KB

Une invitation de commande avec un prompt « (estimateur) » s'ouvre.



La commande « help » donne toutes les commandes disponibles.

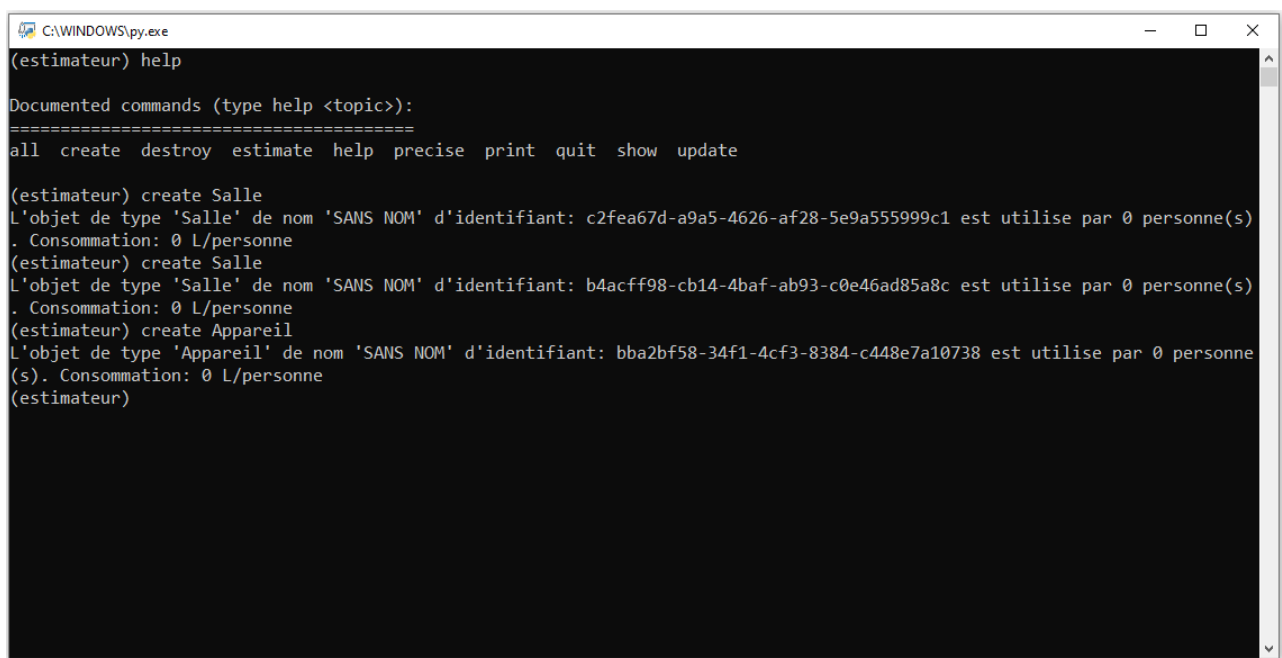


```
C:\WINDOWS\py.exe
(estimateur) help

Documented commands (type help <topic>):
=====
all create destroy estimate help precise print quit show update

(estimateur)
```

Pour le besoin de notre exemple, nous devons créer deux salles (douche et cuisine) et un appareil (machine à laver) en utilisant la commande « create ».



```
C:\WINDOWS\py.exe
(estimateur) help

Documented commands (type help <topic>):
=====
all create destroy estimate help precise print quit show update

(estimateur) create Salle
L'objet de type 'Salle' de nom 'SANS NOM' d'identifiant: c2fea67d-a9a5-4626-af28-5e9a555999c1 est utilise par 0 personne(s).
Consommation: 0 L/personne
(estimateur) create Salle
L'objet de type 'Salle' de nom 'SANS NOM' d'identifiant: b4acff98-cb14-4baf-ab93-c0e46ad85a8c est utilise par 0 personne(s).
Consommation: 0 L/personne
(estimateur) create Appareil
L'objet de type 'Appareil' de nom 'SANS NOM' d'identifiant: bba2bf58-34f1-4cf3-8384-c448e7a10738 est utilise par 0 personne
(s). Consommation: 0 L/personne
(estimateur)
```

Il faut maintenant préciser le nom, le nombre d'utilisateur et la consommation en eau en litre par personne de chaque appareil et salle.

```
C:\WINDOWS\py.exe
(estimateur) update Salle c2fea67d-a9a5-4626-af28-5e9a555999c1 nom "Douche"
Mise à jour de l'attribut 'nom' de l'objet Salle.c2fea67d-a9a5-4626-af28-5e9a555999c1 avec la valeur Douche.
(estimateur) update Salle b4acff98-cb14-4baf-ab93-c0e46ad85a8c nom "Cuisine"
Mise à jour de l'attribut 'nom' de l'objet Salle.b4acff98-cb14-4baf-ab93-c0e46ad85a8c avec la valeur Cuisine.
(estimateur) update Appareil bba2bf58-34f1-4cf3-8384-c448e7a10738 nom "Machine_a_laver"
Mise à jour de l'attribut 'nom' de l'objet Appareil.bba2bf58-34f1-4cf3-8384-c448e7a10738 avec la valeur Machine_a_laver.
(estimateur) update Salle c2fea67d-a9a5-4626-af28-5e9a555999c1 user "6"
Mise à jour de l'attribut 'user' de l'objet Salle.c2fea67d-a9a5-4626-af28-5e9a555999c1 avec la valeur 6.
(estimateur) update Salle b4acff98-cb14-4baf-ab93-c0e46ad85a8c user "6"
Mise à jour de l'attribut 'user' de l'objet Salle.b4acff98-cb14-4baf-ab93-c0e46ad85a8c avec la valeur 6.
(estimateur) update Appareil bba2bf58-34f1-4cf3-8384-c448e7a10738 user "6"
Mise à jour de l'attribut 'user' de l'objet Appareil.bba2bf58-34f1-4cf3-8384-c448e7a10738 avec la valeur 6.
(estimateur) update Salle c2fea67d-a9a5-4626-af28-5e9a555999c1 cons "8"
Mise à jour de l'attribut 'cons' de l'objet Salle.c2fea67d-a9a5-4626-af28-5e9a555999c1 avec la valeur 8.
(estimateur) update Salle b4acff98-cb14-4baf-ab93-c0e46ad85a8c cons "5"
Mise à jour de l'attribut 'cons' de l'objet Salle.b4acff98-cb14-4baf-ab93-c0e46ad85a8c avec la valeur 5.
(estimateur) update Appareil bba2bf58-34f1-4cf3-8384-c448e7a10738 cons "10"
Mise à jour de l'attribut 'cons' de l'objet Appareil.bba2bf58-34f1-4cf3-8384-c448e7a10738 avec la valeur 10.
(estimateur)
(estimateur) all
L'objet de type 'Salle' de nom 'Douche' d'identifiant: c2fea67d-a9a5-4626-af28-5e9a555999c1 est utilise par 6 personne(s).
Consommation: 8 L/personne
L'objet de type 'Salle' de nom 'Cuisine' d'identifiant: b4acff98-cb14-4baf-ab93-c0e46ad85a8c est utilise par 6 personne(s)
. Consommation: 5 L/personne
L'objet de type 'Appareil' de nom 'Machine_a_laver' d'identifiant: bba2bf58-34f1-4cf3-8384-c448e7a10738 est utilise par 6
personne(s). Consommation: 10 L/personne
(estimateur)
```

Notons qu'avec le commande « update » il faut utiliser impérativement soit « nom » pour son nom, soit « user » pour le nombre d'utilisateur, soit « cons » pour consommation, car ce sont les noms de nos attributs dans notre programme. De plus, leur valeur doit être dans des « ' ' » ou « " " ». La commande « all » nous permet de visualiser toutes nos données entrées et sauvegardées.

Pour avoir l'estimation totale de la consommation en eau de notre système, il suffit juste de taper « estimate ».

```
(estimateur) estimate
La consommation totale de tout le systeme est : 138.00 L
(estimateur)
```

Pour avoir l'estimation totale de la consommation en eau des salles ou appareils, il faut utiliser la commande « estimate » avec Salle ou Appareil.






```
(estimateur)
(estimateur) estimate Salle
La consommation totale de tous les 'Salle' est : 78.00 L
(estimateur) estimate Appareil
La consommation totale de tous les 'Appareil' est : 60.00 L
(estimateur)
```

Pour avoir l'estimation totale de la consommation en eau de la machine à laver et de la cuisine, il faut utiliser la commande « precise » avec l'identifiant de la machine à laver et

l'identifiant de la cuisine. Pour connaître leur identifiant, il suffit d'utiliser la commande « all » et de le copier (avec ctrl + ins). La combinaison pour coller sur la console est : shift + ins.

```
(estimateur) precise bba2bf58-34f1-4cf3-8384-c448e7a10738 b4acff98-cb14-4baf-ab93-c0e46ad85a8c
La consommation totale des objets précisés est : 90.00 L
(estimateur)
```

Après utilisation, les données sont sauvegardées dans le fichier « sauvegarde.json » dans le dossier où nous nous trouvons.

	models	4/16/2024 3:16 PM	File folder	
	.gitignore	4/15/2024 11:54 AM	Text Document	1 KB
	console.py	4/16/2024 5:58 PM	Python File	8 KB
	README.md	4/16/2024 5:43 PM	MD File	2 KB
	sauvegarde.json	4/16/2024 9:25 PM	JSON File	1 KB

En relançant le programme puis en entrant la commande « all », nous pouvons constater que nos anciennes données sont toujours présentes et nous pouvons ajouter d'autres si besoin.

```
C:\WINDOWS\py.exe
(estimateur) all
L'objet de type 'Salle' de nom 'Douche' d'identifiant: c2fea67d-a9a5-4626-af28-5e9a555999c1 est utilise par 6 personne(s).
Consommation: 8 L/personne
L'objet de type 'Salle' de nom 'Cuisine' d'identifiant: b4acff98-cb14-4baf-ab93-c0e46ad85a8c est utilise par 6 personne(s)
. Consommation: 5 L/personne
L'objet de type 'Appareil' de nom 'Machine_a_laver' d'identifiant: bba2bf58-34f1-4cf3-8384-c448e7a10738 est utilise par 6
personne(s). Consommation: 10 L/personne
(estimateur)
```

## **CONCLUSION**

En conclusion, le développement de notre outil d'estimation de la consommation d'eau représente une avancée significative dans la gestion responsable de cette précieuse ressource. À travers ce projet, nous avons réussi à concevoir un outil fonctionnel, flexible et convivial, capable d'estimer la consommation d'eau aussi bien à l'échelle domestique qu'industrielle.

Nous avons souligné l'importance de l'utilisation d'outils tels que Visual Studio Code pour faciliter le processus de développement et garantir la qualité du code produit. En outre, nous avons implémenté des fonctionnalités de base telles que la création, l'affichage, la mise à jour, la sauvegarde, et la suppression des objets, tout en offrant une interface en ligne de commande pour interagir avec le programme.

Par ailleurs, nous avons enrichi notre outil en ajoutant des fonctionnalités avancées, notamment une estimation précise de la consommation d'eau pour des objets spécifiques, offrant ainsi une flexibilité et une précision accrues dans son utilisation.

En résumé, ce projet nous a permis de mettre en pratique nos compétences en programmation et en conception logicielle pour créer un outil fonctionnel répondant efficacement aux besoins de gestion de la consommation d'eau, contribuant ainsi à une utilisation plus efficiente et responsable de cette ressource vitale.