Challenge du 15 octobre 2023

@bucdany

4 novembre 2023

La liste de contacts

(Extractions de données dans un texte)

****

1 Énoncé du challenge

Aie ! Je ne retrouve plus ma liste de contact !

Le but de ce challenge est d'extraire les noms, prénoms, téléphones et emails à partir d'un paragraphe, puis de créer une base de données afin de les afficher clairement à l'écran.

Étapes

1. Trouver le moyen de repérer les différentes longueurs et bornes / limites d'extractions.
2. Ranger les données dans une base de données suivant cet ordre : `Nom & Prénom - Mail - Numéro`.
3. Afficher le résultat formaté.

Conditions

- L'affichage se fait via la console.
- Vous êtes libre d'utiliser (ou pas) toutes les librairies de votre choix.
- Vous pouvez coder votre programme aussi bien en fonctionnel qu'en orienté objet.
- L'affichage des données doit respecter exactement le format visuel suivant (voir ci-dessous dans résultat).
- Utilisez les 3 paragraphes ci-dessous pour procéder à l'extraction.

Texte depuis lequel extraire les données

Récemment, j'ai eu l'opportunité de rencontrer des entrepreneurs exceptionnels lors d'une conférence. Par exemple, j'ai été inspiré par l'histoire de M. Thomas Bernard, qui a démarré sa propre entreprise dans la Silicon Valley. Vous pouvez le contacter à l'adresse email thomas.b@alphamail.com ou au numéro suivant +33 1 12 34 56 78. Une autre personne fascinante était Mme Claire Martin, la fondatrice d'une startup technologique innovante. Elle est joignable à cmartin@betainbox.org, son numéro de téléphone est le 09 01 23 45 67. Ensuite, il y avait monsieur Lucas Petit, un innovateur dans le domaine de la construction durable, contactable à lp@experimentalpost.net, son téléphone est le 0890 12 34 56.

En parcourant mon ancien annuaire, je suis tombé sur quelques contacts intéressants. Par exemple, j'ai redécouvert le contact de Mlle Sophie Martin. Son numéro de téléphone est 07 89 01 23 45, et elle est facilement joignable à l'adresse sophie@prototypemail.com. Un autre contact noté était celui du Dr Lucas Dupont. Je me souviens avoir eu plusieurs discussions avec lui. Son numéro est 06 78 90 12 34 et son mail est drdupont@randominbox.org. C'est fascinant de voir comment certains contacts peuvent rapidement nous rappeler des souvenirs passés.

Lors de notre dernière réunion, Madame Jennifer Laroche, joignable au 05.67.890.123 ou par e-mail à laroche@trialmail.net, a exprimé sa satisfaction concernant les avancées du projet. Elle a insisté sur la pertinence du feedback fourni par M. Sébastien Girard, qui peut être contacté au 0456789012 ou par email à sebastieng@demomail.org. De plus,notre consultante externe, mademoiselle Chloé Lefebvre, dont le numéro est le 03.45.67.89.01 et l'e-mail est lefebvre.chloe@testinbox.net, a fourni un rapport détaillé qui a été bien reçu par l'équipe.

Résultat attendu

Nom & Prénom	Mail	Numéro
Thomas Bernard	thomas.b@alphamail.com	33.1.12.34.56.78
Claire Martin	cmartin@betainbox.org	09.01.23.45.67
Lucas Petit	lp@experimentalpost.net	08.90.12.34.56
Sophie Martin	sophie@prototypemail.com	07.89.01.23.45
Lucas Dupont	drdupont@randominbox.org	06.78.90.12.34
Jennifer Laroche	laroche@trialmail.net	05.67.89.01.23
Sébastien Girard	sebastieng@demomail.org	04.56.78.90.12
Chloé Lefebvre	lefebvre_chloe@testinbox.net	03.45.67.89.01

2 Solution

```python
"""
Solution proposée par @bucdany.
Fil de discussion du challenge sur Docstring:
https://discord.com/channels/396825382009044994/1168757567200165940
"""
from re import compile

FILE = "text.txt"
REGEX = (compile(r) for r in (
        r'(?:monsieur|Madame|mademoiselle|Mme|Mlle|M.|Dr)\s((?:[A-Z]\w+\s?){2})',
        r'[\w\.]+@\w+\.\w{2,}',
        r'[+0]\d+'
        ))
PREFIX = ("monsieur", "Madame", "mademoiselle", "Mme", "Mlle", "M.", "Dr")
TITLE = ("Nom & Prénom", "Mail", "Numéro")


def printl(func):
    def wrapper(*args, **kwargs):
        result = func(*args, **kwargs)
        print("\n|" + result, end="")
        return result
    return wrapper


def get_text(file: str) -> str:
    with open(file, encoding="utf8") as f_content:
        return f_content.read()


def format_tel(tel: str) -> str:
    tel_ = "".join(tel[i:i+2] + "." \
    for i in range(0, len(tel), 2)).rstrip(".")
    return tel_[1:2] + tel_[3:1:-1] + tel_[4:] \
    if tel_.startswith("+") else tel_


def extract_regex(txt: str) -> dict:
    return {t: next(REGEX).findall(txt) if i < 2 \
    else list(map(format_tel, next(REGEX).findall("".join(
        t for t in txt if t not in " ."")))) \
        for i, t in enumerate(TITLE)}
```

```
46
47
48  def extract_no_regex(txt: str) -> dict:
49      bdd = {k: [] for k in TITLE}
50
51      txt_ = "".join(t for i, t in enumerate(txt) \
52      if t not in " ." or not txt[i-1].isdigit()).split()
53      for j, l in enumerate(txt_):
54          if l in PREFIX:
55              bdd[TITLE[0]].append(
56                  " ".join(txt_[j+1:j+3])[:-1])
57          elif "@" in l:
58              bdd[TITLE[1]].append(l.rstrip(",").rstrip("."))
59          elif any(n.isdigit() for n in l):
60              bdd[TITLE[2]].append(format_tel(l[:12] \
61              if l.startswith("+") else l[:10]))
62      return bdd
63
64
65  @printl
66  def t_title(*args):
67      return "".join(f'{t:^{length}}|' for t, length \
68      in zip(TITLE, args[0]))
69
70
71  @printl
72  def t_sep_line(*args):
73      return "".join(f'{"-"*length}|' for length in args[0])
74
75
76  @printl
77  def t_content(*args):
78      return "\n|".join("".join(f" {el:<{length-1}}|" \
79      for el, length in zip(el_, args[0])) \
80      for el_ in zip(*args[1].values()))
81
82
83  def show(bdd: dict) -> None:
84      length = [len(max(bdd, key=len))+2 \
85      for bdd in bdd.values()]
86
87      for text_function in [t_title, t_content]:
88          text_function(length, bdd)
89          t_sep_line(length)
90
91
92  if __name__ == "__main__":
93      DATA = get_text(FILE)
```

```
94
95      show(extract_regex(DATA))
96      show(extract_no_regex(DATA))
```

3 Explications du code

Extraction des données

Il y a plusieurs façons de s'y prendre, soit à l'aide des *regex* ou soit dans une logique *builtin* de Python grâce à quelques boucles et un peu de logiques.

Sans *regex*

Tout se passe dans la fonction :

`extract_no_regex(txt: str)-> dict`

On déclare déjà le moyen de stockage des données -> un dictionnaire `bdd` qui contiendra les clefs `("Nom & Prénom", "Mail", "Numéro")`.

L'idée est de tout d'abord nettoyer les données pour permettre d'avoir des choses homogènes.
Les numéros de téléphones sont en effet dans des formats très exotiques avec des points, des espaces et au nombre de chiffres différents en fonction d'un numéro contenant le préfixe international ou pas.

Toutes les extractions sont centrées dans une seule boucle :

`for j, l in enumerate(txt_)`

Les **noms** et **prénoms** sont extraits au travers de cette condition :

```
1 if l in PREFIX:
2   bdd[TITLE[0]].append(" ".join(txt_[j+1:j+3])[:-1])
```

Si l'on trouve un `PREFIX ("monsieur", "Madame", "mademoiselle", "Mme", "Mlle", "M.", "Dr")` alors on récupère les 2 mots suivants que sont nom et prénom.

Les **emails** sont extraits en cherchant un « `@` » :

```
1 elif "@" in l:
2   bdd[TITLE[1]].append(l.rstrip(",").rstrip("."))
```

De plus on nettoie le dernier caractère qui peut contenir un point ou une virgule.

Pour finir l'extraction des `numéros de téléphone` se fait via :

```
elif any(n.isdigit() for n in l):
  bdd[TITLE[2]].append(format_tel(l[:12] \
  if l.startswith("+") else l[:10]))
```

On cherche une chaine de caractères contenant au moins un chiffre avec la fonction `any()` puis on récupère 12 éléments s'il y a un « + » ou bien 10 chiffres seulement.

Notez qu'au préalable, les données sont nettoyées en enlevant les points et les espaces des séries de chiffres des numéros de téléphone :

`...if t not in " ." or not txt[i-1].isdigit()...`

Au moyen des *regex*

Je vous invite à tester vos *regex* via ce site, sans oublier de cocher Python (à gauche) : `https://regex101.com/`

La fonction qui permet de prendre en compte les regex se trouve ici, en une seule ligne `return` :

```
def extract_regex(txt: str)-> dict:
    return {t: next(REGEX).findall(txt) if i < 2 \
    else list(map(format_tel, next(REGEX).findall(
         "".join(t for t in txt if t not in " ."))))  \
    for i, t in enumerate(TITLE)}
```

Les *regex* sont tout d'abord compilées puis utilisée avec `re.findall()` afin de pouvoir tout extraire d'un coup et éviter de surchager le code de boucles inutiles.

La compilation se fait directement à la déclaration des constantes, via l'utilisation d'un simple générateur :

```
REGEX = (compile(r) for r in (
              r'(?:monsieur|Madame|mademoiselle|Mme|Mlle|M.
                 |Dr)\s((?:[A-Z]\w+\s?){2})',
              r'[\w.]+@\w+\.\w{2,}',
              r'[+0]\d+'))
```

Pour plus d'information sur la compilation des *regex* : `https://pynative.com/python-regex-compile/`

Les **noms** et **prénoms** sont extraits par cette *regex* :

`r'(?:monsieur|Madame|mademoiselle|Mme|Mlle|M.|Dr)\s((?:[A-Z]\w+\s?){2})`

- un premier bloc qui me permet de chercher la civilité de la personne : `?:monsieur|Madame|mademoiselle|Mme|Mlle|M.|Dr)`

notez le `?:` qui permet d'utiliser des parenthèses associatives sans en extraire le contenu. Ainsi le résultat de la recherche n'enverra pas ces données.
- le `\s` pour chercher un espace.
- un jeu de parenthèses pour permettre d'extraire proprement les données et d'éviter aussi des répétitions dans la *regex* :
`((?:[A-Z]\w+\s?){2})`

Ainsi, on cherche ici ce pattern :

```
(?:[A-Z]\w+\s?){2}
```

qui est donc répété 2x via `{2}`. Notez encore un `?` qui permet de définir un caractère optionnel, ici l'espace noté `\s` qui est présent entre les noms et prénoms mais pas à la fin de la chaine à extraire. Avec `[A-Z]` on cherche une majuscule puis un mot avec `\w+` et finalement un espace `\s`. Encore une fois le `?:` permet d'éviter d'avoir des doublons dans le résultat de la *regex*.

Les `emails` sont extrait grâce à cette *regex* :

```
r'[\w.]+@\w+\.\w{2,}'
```

- on cherche un mot pouvant contenir un point
- le caractère « `@` »
- un autre mot derrière
- un point
- un mot de 2 lettres minimum pour l'extension de l'adresse mail

Pour finir l'extraction des `numéros de téléphone` se fait via cette *regex* :

```
r'[+0]\d+'
```

- on cherche le signe « `+` » ou le chiffre « `0` » `[+0]`
- suivi de plusieurs chiffres `\d+`

Au niveau du code dans mon `return`, j'utilise `next()` qui me permet d'incrément le pointeur du générateur et passer à la *regex* suivante. Ainsi mes deux première extractions se font via :

```
next(REGEX).findall(txt)
```

La dernière, elle, pour les numéros de téléphone se pose sur le texte nettoyé des points et des virgules, grâce à cette boucle :

```
"".join(t for t in txt if t not in " .")
```

Stokage temporaire des données

Avec ou sans *regex*, les fonctions d'extraction retournent toutes les deux un dictionnaire contenant les données suivantes :

```
{'Mail': ['thomas.b@alphamail.com',
          'cmartin@betainbox.org',
          'lp@experimentalpost.net',
          'sophie@prototypemail.com',
          'drdupont@randominbox.org',
          'laroche@trialmail.net',
          'sebastieng@demomail.org',
          'lefebvre_chloe@testinbox.net'],
 'Nom & Prénom': ['Thomas Bernard',
                  'Claire Martin',
                  'Lucas Petit',
                  'Sophie Martin',
                  'Lucas Dupont',
                  'Jennifer Laroche',
                  'Sébastien Girard',
                  'Chloé Lefebvre'],
 'Numéro': ['33.1.12.34.56.78',
            '09.01.23.45.67',
            '08.90.12.34.56',
            '07.89.01.23.45',
            '06.78.90.12.34',
            '05.67.89.01.23',
            '04.56.78.90.12',
            '03.45.67.89.01']}
```

Formatage des données

Les données des numéros téléphoniques sont à reformatées afin de respecter l'énoncer et c'est par l'appel de cette fonction que ça se passe :

```python
def format_tel(tel:str)-> str:
    tel_ = "".join(tel[i:i+2] + "." \
        for i in range(0, len(tel), 2)).rstrip(".")
    return tel_[1:2] + tel_[3:1:-1] + tel_[4:] \
        if tel_.startswith("+") else tel_
```

Les numéros sont tout d'abord formatées dans un premier temps pour les numéros nationnaux commençant par 0 :

```
tel_ = "".join(tel[i:i+2] + "." for i in range(0, len(tel), 2)).rstrip
       (".")
```

On ajoute d'abord un point tous les deux chiffres pour formater un numéro national.

Quant aux numéraux internationaux (commençant par le symbole « + ») alors on doit intervertir un point et un chiffre :

`tel_[3:1:-1]`

En d'autres termes, transformer ce format `+3.31.12.34.56.78` en celui-ci `33.1.12.34.56.78`.

Affichage

Là, c'est une grosse partie et je voudrais pour cela vous présenter, dans un but pédagogique, un peu les décorateurs qui ont leur importance en Python.

L'affiche est géré par la fonction :

`show(bdd: dict)-> None`

qui fait appel à 3 autres fonctions, chacune décorée de `@printl` :

```
def printl(func):
    def wrapper(*args):
        result = func(*args)
        print("|" + result)
        return result
    return wrapper

...

@printl
def t_title(*args):
    return "".join(f'{t:^{length}}|' for t, length \
        zip(TITLE, args[0]))

@printl
def t_sep_line(*args):
    return "".join(f'{"-"*length}|' for length in args[0])

@printl
def t_content(*args):
    return "\n|".join("".join(f" {el:<{length-1}}|" \
        for el, length in zip(el_, args[0])) \
        for el_ in zip(*args[1].values()))

def show(bdd: dict)-> None:
    length = [len(max(bdd, key = len))+2 \
    for bdd in bdd.values()]

    for text_function in [t_title, t_content]:
        text_function(length, bdd)
        t_sep_line(length)
```

En termes simples, un décorateur permet d'encapsuler une fonction, qu'il prend en paramètre, pour intégrer d'autres actions autour d'elle.

```
def printl(func):
    def wrapper(*args):
        result = func(*args)
        print("|" + result)
        return result
    return wrapper
```

Ici, je fais simplement un `print("|" + result)`, cela me permet de simplifier la fonction `show()` et les formatages des contenus du tableau.

`*args` permet de récupérer les arguments de la fonction et si besoin de les utiliser dans le décorateur. Ici j'affiche simplement ce que me renvoi les fonctions en ajoutant un « | » au tout début.

On utilise les décorateur 1 ligne au-dessus des fonctions, Python s'occupe du reste.

```
@printl
def t_title(*args):
    return "".join(f'{t:^{length}}|' for t, length \
        in zip(TITLE, args[0]))
```

l'appel de cette fonction fait donc simplement cela :

```
print("|" + "".join(f'{t:^{length}}|' for t, length
    in zip(TITLE, args[0])))
```

Ce serait donc équivalent à cela sans le décorateur :

```
def t_title(length:list)-> None:
    print("|" + "".join(f'{t:^{l}}|' for t, l \
    in zip(TITLE, length)))
```

Au passage, cette fonction réunie les titres avec la taille des colonnes grâce à la fonction `zip()` puis on centre le tout avec les fonctions de formatage de `fstring` : « :^ » (pour centrer).

Pour plus d'info, voici un lien pour mieux appréhender `fstring` :
`https://discord.com/channels/396825382009044994/1155460501409628230`

La deuxième fonction pour les lignes de séparation fonctionne de la même façon :

```
@printl
def t_sep_line(*args):
    return "".join(f'{"-"*length}|' for length in args[0])
```

On affiche des `"-"` en fonction de la taille des colonnes.

Pour finir le contenu du tableau se fait par l'intermédiaire de 2 boucles et pour le coup 2x `zip()` (oui j'aime beaucoup les `zip`) :

```
@printl
def t_content(*args):
    return "\n|".join("".join(f" {el:<{length-1}}|" \
    for el, length in zip(el_, args[0])) \
    for el_ in zip(*args[1].values()))
```

- La première boucle :
  `for el_ in zip(*args[1].values())`
  permet de lire chaque ligne du contenu du tableau.
- la deuxième, permet, comme dans la fonction des titres, de créer un tuple pour associer les tailles de colonnes du tableau.
- Le formatage permet d'aligner à gauche : « `:<` » et ajoute des espaces pour bien formater les données dans les cases.

Pour revenir à la fonction `show()` :

```
def show(bdd: dict)-> None:
    length = [len(max(bdd, key = len))+2 \
    for bdd in bdd.values()]

    for text_function in [t_title, t_content]:
        text_function(length, bdd)
        t_sep_line(length)
```

La taille de chaque colonne est calculé par l'intermédiaire de `max(..., key = len)` qui est très pratique pour éviter de faire encore des boucles, ainsi l'élément de la liste de taille la plus longue est alors directement renvoyée, et c'est la fonction `len()` qui permet de récupérer la taille en question. `length` est donc une liste contenant la taille de chaque colonne du tableau.

Cette dernière fonctionnalité est un peu bonus pour éviter les répétition de la ligne séparatrice du tableau :

```
    for text_function in [t_title, t_content]:
        text_function(length, bdd)
        t_sep_line(length)
```

J'itère donc sur les deux types d'éléments à ajouter dans le tableau : `t_title` et `t_content`, qui sont appelés avec les arguments `length` et `bdd`.

Puis après chaque type d'élément du tableau, une ligne séparatrice est ajoutée : `t_line()`.