Série 1 - Jupyter - votre premier "Notbook Python"

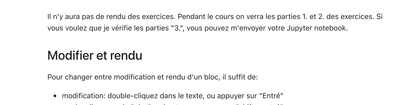
Ce système vous permet de facilement utiliser le langage Python dans un navigateur web. Il permet d'offrir un environnement facile à utiliser (mais sans correcteur d'orthographe :). On va y aller doucement: chaque exercice consiste de trois parties (encore Bloom...):

1. *Connaissance* - Une introduction avec un bout de Python qui tourne toute seule
2. *Compréhension* - Un ou plusieurs exercices ou ces mêmes commandes sont à utiliser par vous-mêmes
3. *Application+* - Un ou plusieurs exercices qui requièrent des commandes supplémentaires ou même la programmation

Concernant le cours, et surtout l'examen, je fais les exigeances suivantes:

1. Il faut avoir compris cela - il y aura des questions dessus
2. Si vous n'arrivez pas à faire ces exercices, vous devrez réfléchir sur le choix d'enseignant informatique ;) Mais c'est surtout pour approfondir la compréhension
3. Ceci est un bonus. Pour ceux qui connaissent déjà la programmation. Il pourrait avoir des questions bonus à l'examen, mais vous arrivez à la note maximale sans ces questions.

Enregistrement et Récupération



- Download - qui vous donnera une copie de votre feuille de travail que vous pouvez enregistrer sur votre ordinateur
- Upload - mettre un fichier depuis votre ordinateur sur le Jupyter - vous pouvez aussi glisser/déposer votre fichier dans la partie gauche
- Navigateur - enregistrer et récupérer depuis votre navigateur. Seulement une version va être gérée par votre navigateur!

Rendu d'exercices

Il n'y aura pas de rendu des exercices. Pendant le cours on verra les parties 1. et 2. des exercices. Si vous voulez que je vérifie les parties "3.", vous pouvez m'envoyer votre Jupyter notebook.

Modifier et rendu

Pour changer entre modification et rendu d'un bloc, il suffit de:

- modification: double-cliquez dans le texte, ou appuyer sur "Entré"
- rendu: cliquer sur le "play" en haut, ou appuyez sur "shift + entré"

Navigation

Vous pouvez utiliser la navigation avec les curseurs "Up" et "Down", mais aussi avec la souris, bien sûr... Pour naviguer à l'intérieur d'une ligne, vous pouvez utiliser la touche ALT (Option sous Mac) plus les curseurs "Droite" et "Gauche" pour sauter d'un mot à l'autre. C'est un peu plus rapide... Bien sûr que les "PgUp", "PgDn", "Home", et "End" marchent aussi.

Exercice 2

Ce deuxième exercice va se produire dans une partie `code` où vous devez écrire quelques lignes de Python! Mais pas de soucis, on va y aller progressivement. En-dessous de ce block de texte vous trouverez trois blocks pour les trois niveaux de l'exercice.

On va se pencher sur la *Differential Privacy* et faire quelques exercices dessus.

1. Connaissance

Dans la première partie vous trouvez une petite fonction qui prend comme entrée si vous êtes un délinquant, et qui sort une réponse protégée par la *Differential Privacy*. si vous faites tourner le code, il vous donnera quelques réponses pour des entrées différentes.

Vous pouvez faire tourner le block plusieurs fois, et il devrait vous afficher des résultats différents presque chaque fois.

2. Compréhension

Générateurs aléatoire

Pourquoi en lançant le block plusieurs fois vous recevez des résultats différents *presque* chaque fois?

Espérance mathématique

On va essayer de trouver l'espérance mathématique de notre fonction dépendant si on est innocent ou pas. Au lieu de le faire mathématiquement, on va le faire par essai et contage, et un peu de bon sens...

Dans le block `Exercice 2 - Partie 2`, ajoutez 10 fois la ligne suivante:

```
print_guilty(dp_1(True))
```

puis lancez le block.

- Combien de fois vous trouvez `guilty`, combien de fois `innocent`?
- Quelle est donc l'espérance mathématique si on met `guilty` à `1`, et `innocent` à `0`?
- La même question, mais si on met `print_guilty(dp_1(false))`

Correction de la DP

- Supposons qu'on a seulement une personne qui est coupable - combien de coupables va-t-on trouver en moyenne?
- En connaissant l'espérence mathématique de `dp_1(False)`, comment on peut calculer la valeur probable de personnes coupables?

3. Application

Si vous connaissez un peu la programmation, alors on peut faire les calculs un peu plus correcte.

Créer un nombre élevé de mesures

La première méthode `create_measures` va remplacer notre utilisation ligne par ligne d'appel à `dp_1`. Le paramètre `p_guilty` indique la probabilité entre 0 et 1 qu'un élément est coupable.

Calculer le nombre de personnes coupables

La deuxième méthode `calculate_guilty` prend la sortie de `create_measures` pour calculer le nombre probable de personnes coupables. Il faudra d'abord compter le nombre de `True` dans l'entré, puis le mettre en relation avec le nombre total de réponses. Après il faut corriger par rapport à l'erreur introduite par la DP.

In [1]:
```
# Exercice 2 - Partie 1
import random

# Returns True or False for a coin toss. The random.choice method chooses randomly be
# the two values. Think of "True" as "Tail", and "False" as "Head"
def coin() -> bool:
    return random.choice([True, False])

# Differential Privacy 1 - takes a vairable as input that indicates if the real value
# not. Then it uses DP to decide whether it should output the real value, or a made-u
def dp_1(guilty: bool) -> bool:
    if coin():
        return guilty
    else:
        return coin()

# A pretty-printing method that shows nicely what is going on.
def print_guilty(guilty: bool) -> str:
    if guilty:
        print("Is guilty")
    else:
        print("Is innocent")

# Two outputs for a guilty and an innocent person:
print_guilty(dp_1(True))
print_guilty(dp_1(False))
```

```
Is guilty
Is innocent
```

In [61]:
```
# Exercice 2 - Partie 2

print("10 guilty answers")
print_guilty(dp_1(True))
print_guilty(dp_1(True))
print_guilty(dp_1(True))
print_guilty(dp_1(True))
print_guilty(dp_1(True))
print_guilty(dp_1(True))
print_guilty(dp_1(True))
print_guilty(dp_1(True))
print_guilty(dp_1(True))
print_guilty(dp_1(True))
# 1 innocent, 9 guilty
# Espérance mathématique ici: 0.9 - en suivant le schéma: 0.75

print("10 innocent answers")
print_guilty(dp_1(False))
print_guilty(dp_1(False))
print_guilty(dp_1(False))
print_guilty(dp_1(False))
print_guilty(dp_1(False))
print_guilty(dp_1(False))
print_guilty(dp_1(False))
print_guilty(dp_1(False))
print_guilty(dp_1(False))
print_guilty(dp_1(False))
# 6 innocent, 4 guilty
# Espérance mathématique ici: 0.4 - en suivant le schéma: 0.25
```

```
10 guilty answers
Is guilty
Is guilty
Is guilty
Is guilty
Is guilty
Is guilty
Is innocent
Is guilty
Is guilty
Is guilty
10 innocent answers
Is guilty
Is innocent
Is innocent
Is guilty
Is innocent
Is innocent
Is guilty
Is guilty
Is innocent
Is innocent
```

Corriger la DP

1 personne coupable -> en moyenne

- coupable: Espérance_coupable = 0.5 + 0.5 * 0.5 = 0.75
- innocents: Espérance_coupable = 9 0.5 0.5 = 2.25
- total: 0.75 + 2.25 = 3 personnes coupables en moyenne

Corriger le résultat

Sur `n` résultats, il y a en moyenne

- `n/4` qui sont aléatoirement coupable
- `n/4` qui sont aléatoirement innocent

Donc si on a `total_coupable` et `total_innocent` de cas, on peut déduire les cas aléatoire en calculant:

- `vraiment_coupable = total_coupable - n / 4`
- `vraiment_innocent = total_innocent - n / 4`

Et donc pour calculer la distribution des personnes coupables, on peut calculer:

$$P(\text{coupable}) = \frac{\text{vraiment\_coupable}}{n / 2}$$

In [85]:
```
# Exercice 2 - Partie 3

# This method returns a number of throws where each throw is randomly chosen to be
# from a guilty person with probability p_guilty.
# The return value should be an array of booleans.
def create_measures(throws: int, p_guilty: float) -> [bool]:
    dp = []
    for choice in random.choices([True, False], [p_guilty, 1-p_guilty], k=throws):
        dp.append(dp_1(choice))

    return dp

# Returns the most probable number of guilty persons given the array of results.
def calculate_guilty(results: [bool]) -> float:
    total = len(results)
    total_guilty = len(list(filter(lambda x: x, results)))
    total_innocent = len(list(filter(lambda x: not x, results)))
    return (total_guilty - total / 4) / (total / 2)

# This should print a number close to 0.1 * 100 = 10 guilty persons.
print(f"The percentage of guilty persons are: {calculate_guilty(create_measures(10000,
```

```
The percentage of guilty persons are: 49.18
```

In [ ]:
```

```