Ajouter une date de réception aux ordinateurs

/home/phil/Documents/LMG/STS_A_2025-2027/PYTHON_STS2/01_eval2_GESTION_PARC_INFORMATIQUE_

Table des matières

1. Contexte....1
2. Faire évoluer le modèle d'un PC....1
3. Comment représenter une date en Python ?....2
   a. La classe date....2
   b. La classe datetime....2
   c. Obtenir la date du jour....3
4. Modifier le constructeur de PC....3
   a. Créer un PC avec une date de réception et afficher l'information....4
   b. Utiliser une date de réception par défaut....4
   c. À retenir sur les paramètres par défaut....5
5. Défi – Calculer l'ancienneté du PC et plus encore....6
   a. À retenir sur l'utilisation de dates....6

1. Contexte

Le lycée possède plusieurs dizaines d'ordinateurs répartis dans différentes salles. Pour assurer le suivi du parc informatique, le responsable informatique souhaite connaître non seulement les caractéristiques de chaque PC, mais également **la date à laquelle chaque ordinateur a été reçu par le lycée**.

Cette information permettra notamment de répondre à des questions telles que :

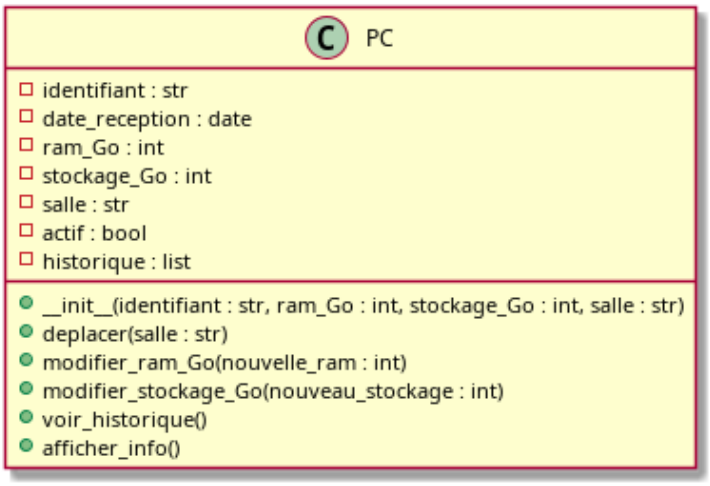
- Quand cet ordinateur a-t-il été reçu ?
- Quels ordinateurs ont été reçus récemment ?
- Depuis combien de temps un PC fait-il partie du parc informatique ?

Il est donc nécessaire de faire évoluer la classe `PC`.

2. Faire évoluer le modèle d'un PC

Nous allons ajouter une nouvelle propriété : `date_reception`.

Cette propriété contiendra **la date à laquelle le PC a été reçu au lycée**.



Question de reflexion :

Comment stocker une date ? Quelles difficultés peut-on rencontrer avec ce type d'information?

3. Comment représenter une date en Python ?

Python possède une bibliothèque standard appelée `datetime`.

Elle permet notamment de manipuler :

- des dates ;
- des heures ;
- des dates et heures ;
- des durées.

Pour utiliser cette bibliothèque, on peut écrire : `import datetime`

Pour obtenir la date du jour il suffit de taper : `datetime.date.today()`

Cependant, Python propose également une écriture permettant d'importer directement certains éléments de cette bibliothèque.

Par exemple : `from datetime import date`

On peut alors utiliser directement la classe `date` et obtenir la date du jour en tapant date.today()

Vérifier cela avec deux mini-programmes :

import datetime aujourdhui = datetime.date.today() print(aujourdhui) print(type(aujourdhui))	from datetime import date aujourdhui = date.today() print(aujourdhui) print(type(aujourdhui))

a. La classe `date`

Une date Python peut être créée à partir de trois informations :

```
date(année, mois, jour)
```

Par exemple :

```
from datetime import date

date_reception = date(2026, 9, 15)
```

La variable `date_reception` contient alors la date : `2026-09-15`

À retenir

Une `date` représente uniquement : `année + mois + jour` Elle ne contient pas d'heure.

b. La classe `datetime`

La bibliothèque `datetime` contient également une classe appelée `datetime`. Elle permet de représenter une date **et une heure**.

Exemple :

```
from datetime import datetime

date_heure = datetime(2026, 9, 15, 14, 30)
```

Cette valeur représente : `15 septembre 2026 à 14 h 30`

On peut donc distinguer :

Type	Contenu
`date`	année, mois, jour
`datetime`	année, mois, jour, heure, minute, seconde

Dans notre application, nous avons uniquement besoin de connaître **le jour de réception du PC**. Nous utiliserons donc `date`.

c. Obtenir la date du jour

La classe `date` permet d'obtenir la date actuelle grâce à la méthode : `date.today()`

Exemple :

```
from datetime import date

aujourd_hui = date.today()
print(aujourd_hui)
```

Le résultat dépendra du jour où le programme est exécuté. Par exemple : `2026-09-21`

4. Modifier le constructeur de PC

La date de réception doit être connue **lors de la création du PC**. Nous allons donc modifier le constructeur.

Avant :

```
def __init__(self, identifiant, ram_Go, stockage_Go, salle):
```

Après :

```
def __init__(self, identifiant, ram_Go, stockage_Go, salle, date_reception):
```

La valeur reçue par le constructeur devra être stockée dans l'attribut :

```
self.date_reception
```

Travail demandé

Modifier votre constructeur afin qu'il puisse recevoir une date de réception. Vous devez obtenir une structure similaire à :

```
from datetime import date

class PC:
    def __init__(self, identifiant, ram_Go, stockage_Go, salle, date_reception):
        self.identifiant = identifiant
        self.ram_Go = ram_Go
        self.stockage_Go = stockage_Go
        self.salle = salle
        self.date_reception = date_reception

        # Suite du constructeur...
```

a. Créer un PC avec une date de réception et afficher l'information

Supposons que le lycée ait reçu un ordinateur le 3 septembre 2026.
On peut créer l'objet ainsi :

```
pc1 = PC( "PC-001", 8, 512, "E03B", date(2026, 9, 3) )
```

La propriété : `pc1.date_reception` contient alors la date : `2026-09-03`

Afficher la date de réception

Nous pouvons accéder à l'attribut comme pour les autres propriétés :

```
print(pc1.date_reception)
```

Python affichera généralement la date sous la forme : `2026-09-03`

Nous pouvons également l'intégrer à la méthode `afficher_info()`.

Ajouter une instruction permettant d'obtenir : `Date de réception : 2026-09-03`

Valider par un test le fonctionnement de la nouvelle version de `afficher_info()`

b. Utiliser une date de réception par défaut

Lorsqu'un nouveau PC arrive au lycée, il est généralement enregistré le jour même de sa réception.
Nous souhaitons donc simplifier la création d'un objet `PC` :

- si une date de réception est fournie, elle doit être utilisée ;
- si aucune date n'est fournie, le PC doit automatiquement recevoir **la date du jour** comme date de réception.

Formulation technique du problème

Actuellement, la création d'un PC nécessite de fournir une date :

```
pc1 = PC("PC-001", 8, 512, "E03B", date(2026, 9, 10))
```

Mais si le PC est reçu aujourd'hui, il serait pratique de pouvoir écrire simplement :

```
pc2 = PC("PC-002", 16, 1000, "F10")
```

Dans ce cas, Python doit automatiquement utiliser la date du jour.

Fonctions et paramètres par défaut

En Python, il est possible de donner une valeur par défaut à un paramètre dans une fonction.
Exemple :

```
def bonjour(nom="visiteur"):
    print("Bonjour", nom)
```

On peut alors écrire : `bonjour("Alice")` ou simplement : `bonjour()`

Dans le deuxième cas, Python utilise la valeur passée en paramètre : `Bonjour Alice`

Dans le deuxième cas, Python utilise la valeur par défaut : `Bonjour visiteur`

Modification du constructeur

Notre constructeur doit maintenant accepter une date de réception **facultative**.

On souhaite pouvoir écrire les deux instructions suivantes :

```
pc1 = PC("PC-001", 8, 512, "E03B", date(2026, 9, 10))
```

et :

```
pc2 = PC("PC-002", 16, 1000, "F10")
```

Modifier la définition du constructeur pour que le paramètre `date_reception` possède la valeur `date.today()` par défaut.

Le constructeur devra avoir une forme proche de :

```
def __init__(self, identifiant, ram_Go, stockage_Go, salle, date_reception=...):
```

Tester votre modification

Créer deux objets :

- Un ordinateur *pc1* reçu le **10 septembre 2026** :
  ```
  pc1 = PC("PC-001", 8, 512, "E03B", date(2026, 9, 10))
  ```
- Un ordinateur *pc2* reçu aujourd'hui :
  ```
  pc2 = PC("PC-002", 16, 1000, "F10")
  ```
- Afficher leur date de réception :
  ```
  print(pc1.date_reception)
  print(pc2.date_reception)
  ```

c. À retenir sur les paramètres par défaut

Un **paramètre par défaut** permet de rendre un argument facultatif lors de l'appel d'une fonction ou d'un constructeur.

```
def fonction(parametre=valeur_par_defaut):
    ...
```

On peut donc avoir plusieurs paramètres avec une valeur par défaut. Mais les paramètres avec une valeur par défaut doivent être placés après les paramètres obligatoires.

def exemple(a=1, b=2, c=3, d=4): print(a, b, c, d)	✅ Correct
def creer_pc(identifiant, ram_Go, stockage_Go, salle, date_reception=None): ...	✅ Correct
def creer_pc(identifiant, date_reception=None, ram_Go, stockage_Go): ...	❌ Incorrect

5. Défi – Calculer l'ancienneté du PC et plus encore...

La présence d'une date de réception permet d'aller plus loin.

On souhaite maintenant savoir depuis combien de jours un PC appartient au parc informatique.

Python permet de soustraire deux objets `date`.

Exemple :

```
from datetime import date

date_reception = date(2026, 9, 3)
aujourd_hui = date.today()

anciennete = aujourd_hui - date_reception
print(anciennete)
```

Le résultat est un objet de type `timedelta`.

Par exemple : `18 days, 0:00:00`

On peut obtenir le nombre de jours grâce à : `anciennete.days`

Travail demandé

- Tester la différence entre deux dates et observer le type des données obtenues (type de anciennete et anciennete.days) . Qu'en pensez-vous ?
- Créer une méthode : `anciennete()` Cette méthode devra calculer et retourner le nombre de jours écoulés depuis la réception du PC. Exemple :

```
Ce PC appartient au lycée depuis 18 jours.
```

Indice

Vous aurez besoin de : `date.today()` et de : `self.date_reception`

- créer une fonction ajouter_historique qui ajoute à l'historique des opérations non seulement le texte fourni mais aussi la date du jour sous la forme d'un couple (date : date , informations : string)
- modifier la fonction voir_historique() pour afficher correctement cette nouvelle liste :

```
2026-09-03  Modification de la Ram : 8 Go  - > 16 Go
2026-09-03  Modification du stockage : 512 Go - > 1000 Go
```

a. À retenir sur l'utilisation de dates

Une date n'est pas nécessairement une chaîne de caractères.

```
self.date_reception = "03/09/2026"  # ❌ Exemple peu adapté

self.date_reception = date(2026, 9, 3) # ✅ Exemple adapté
```

La bibliothèque standard de Python contient un module `datetime.` L'utilisation d'objet *date* permet ensuite d'effectuer des opérations sur les dates, notamment des comparaisons et des calculs de durée. C'est bien plus simple, robuste, fiable, réutilisable et partageable … que l'utilisation d'une méthode personnelle bricolée.