Notions de base Python

Parfait! Voici un **résumé complet et détaillé** des notions de **base Python + POO**, bien structuré pour bien comprendre chaque partie 4



🐍 PYTHON : Notions de base

- 1. Structures de données fondamentales
- Listes (list)
- Collection ordonnée, modifiable, acceptant les doublons
- Création :

```
python
fruits = ['pomme', 'banane', 'orange']
```

- Accès: fruits[0] → 'pomme'
- Fonctions utiles:
 - append(x): ajoute un élément
 - remove(x): supprime un élément
 - sort(): trie la liste
 - reverse(), pop(), insert()
- ✓ Dictionnaires (dict)
- Stocke des paires clé-valeur
- Création :

```
python

personne = {'nom': 'Ali', 'age': 25}
```

Accès: personne['nom']

• Méthodes utiles: get(), keys(), values(), items(), update(), pop()

Tuples (tuple)

- Collection **ordonnée** et **immutable** (non modifiable)
- Utilisés pour les données fixes

```
python

coord = (10, 20)
```

• 2. Boucles et conditions

Boucles

For (itération):

```
python

for i in range(5):
    print(i)
```

While (tant que):

```
python

x = 0
while x < 5:
    print(x)
    x += 1</pre>
```

Conditions

```
python

age = 20
if age < 18:
    print("Mineur")

elif age == 18:
    print("Tout juste majeur")</pre>
```

```
else:

print("Majeur")
```

Opérateurs:

```
• Comparaison: == , != , < , > , <= , >=
```

- Logiques: and, or, not
- Appartenance: in, not in

• 3. Fonctions & Lambda

Fonctions

```
python

def dire_bonjour(nom):
    return f"Bonjour {nom}"
```

• Paramètres par défaut :

```
python

def saluer(nom="inconnu"):
    return f"Bonjour {nom}"
```

- Fonctions Lambda
- Fonctions anonymes (courtes, en une ligne)

```
python

addition = lambda x, y: x + y
print(addition(3, 4)) # 7
```

4. Fonctions d'ordre supérieur: map(), filter(), reduce()

map() - Appliquer une fonction à chaque élément

```
python

nombres = [1, 2, 3]
res = list(map(lambda x: x * 2, nombres)) # [2, 4, 6]
```

filter() - Garder les éléments qui satisfont une condition

```
python
pairs = list(filter(lambda x: x % 2 == 0, nombres)) # [2]
```

reduce() - Réduction d'une liste à une seule valeur

```
from functools import reduce
somme = reduce(lambda x, y: x + y, nombres) # 6
```

5. Modules & Packages

Modules

- Fichier .py contenant des fonctions/classes
- Exemple :

```
python

import math
print(math.sqrt(9)) # 3.0
```

Packages

- Répertoire contenant plusieurs modules
- Nécessite un fichier __init__.py
- Exemple :

```
python
```

```
from mon_package import mon_module
```

Création:

```
markdown
mon_package/
/ __init__.py
 — module1.py
├─ module2.py
```

6. Gestion des Exceptions

Try / Except / Finally

```
python
try:
   x = 10 / 0
except ZeroDivisionError:
    print("Erreur : division par zéro")
finally:
    print("Bloc finally exécuté")
```

- Autres erreurs fréquentes : ValueError , IndexError , KeyError , TypeError
- Lancer une exception personnalisée :

```
python
raise ValueError("Valeur invalide")
```



🌠 Programmation Orientée Objet (POO) en Python

• 1. Classe & Objet

```
class Voiture:
    def __init__(self, marque, annee):
        self.marque = marque
        self.annee = annee

    def klaxonner(self):
        return f"{self.marque} : Bip Bip !"

# Création d'objet
v1 = Voiture("Toyota", 2020)
print(v1.klaxonner())
```

- __init__ : constructeur (appelé automatiquement)
- self : référence à l'objet courant

2. Encapsulation

• Cacher les détails internes

```
class Compte:
    def __init__(self):
        self.__solde = 0 # Attribut privé

def afficher_solde(self):
    return self.__solde
```

• 3. Héritage

```
python
```

```
class Animal:
    def parler(self):
        print("Je fais un son")

class Chien(Animal):
    def parler(self):
        print("Wouf")

rex = Chien()
rex.parler() # Wouf
```

• Classe enfant hérite des attributs et méthodes de la classe parent

4. Polymorphisme

• Même méthode, comportement différent selon l'objet :

```
python

for animal in [Chien(), Animal()]:
    animal.parler()
```

5. Abstraction

Avec abc (abstract base class)

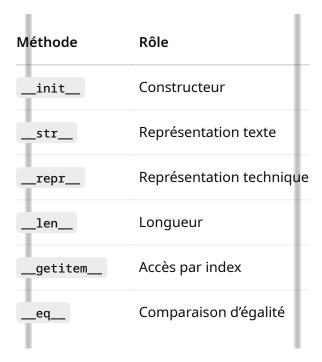
```
python

from abc import ABC, abstractmethod

class Forme(ABC):
    @abstractmethod
    def aire(self):
        pass
```

```
class Cercle(Forme):
    def aire(self):
        return 3.14 * 5 * 5
```

• 6. Méthodes spéciales



Souhaitez-vous que je génère un **PDF téléchargeable** de ce résumé ? Ou voulez-vous une **fiche en une page** pour réviser rapidement ?