Chapitre 0 : Rappel

Séance 1: Notions de base en Python

Anis SAIED

IPEIN

Septembre 2025

Plan de la séance

Table of contents

Python: Rappel général

- Documentation officielle
- Installer Python: Core, standard library
- Open source
- Utilisation: Web, API, Data Science,
- Emplois
- Certifications: 4C, Microsoft

Un programme Python

- EDI: IDLE, Spyder, Pyzo,
- Code source: instructions, blocs, commentaires
- byte code, interpreter
- Executer un programme: mode script
- 3 fichiers standards associés à chaque programme: std Input / std Output / std Error
- Exemple: Le **shell** (programme, execute des lignes de commandes)
  - std input: clavier
  - std output: écran
  - std error: écran
- IDLE > run > shell $\rightarrow$ Mode interatif
- IDLE > new > window/file $\rightarrow$ Mode script

Table of contents

Mots clés

Le choix de nom de variable de doit pas être l'un de ces mots clés.

Python keywords

for, in, range, while, continue, break, yield,
if, elif, else,
or, and, not,
open, close, print, input,
def, global, return, pass, global
lambda, filter, map,
try, except, finally, raise, assert
...

Type de variable

- Non typée
- type change selon son contenu: `type(var)`

Le types: int & float

- Opérateurs: $+, -, *, /$
- taille: limitée à l'espace mémoire disponible

```
n=1 ; n=n+1 ; n *=2 ; type(n) \# int
type(n)==int \# True
```

- Notation scientifique

```
f=1e-2 \#0.01
f=3. type(f)==float \# True
```

Le type bool

- Opérateurs: *and*, *or*, *not*
- Valeurs possibles: `True, False`

  ```
  a=0 ; b=1; c=0. ; d=-1.
  ```
- *bool*(*expr*) `evalue logiquement une expression.`
- `Toute variable non nulle/vide est évaluée comme True.`

  ```
  bool(a) ; bool(b) \#False True
  ```

Le type complex

- partie imaginaire, partie réelle

Conversion de types

- $\boxed{dir(int)}$ retourne les fonctions disponibles et applicables sur les valeurs de ce type.
- Convertir une valeur de type A en type B permet de profiter des fonctions offertes par le type B

  `a=1.0 ; a = int(a)` → passer de `float à int`

  `a = int("abc")` → `Erreur de conversion de type`
- `Rappel : Toute erreur arrête l'exécution du programme → Il faut prévoyer un traitement alternatif en cas d'erreur`

Gestion d'erreurs

- Toute instruction qui ne respecte pas les règles du langage Python est considérée comme une **Exception**
- Une Exception entraine l'arrêt immédiat du programme
- Eviter l'arrêt imprévisible du programme necessite
  1. Ecrire les instrcutions succeptibles d'arrêter le programme dans le bloc `try`
  2. Ecrire le traitement alternatif (qui corrige cette erreur) dans le bloc `except`
- Poursuivre la suite du programme après la sortie du bloc `try ... except`

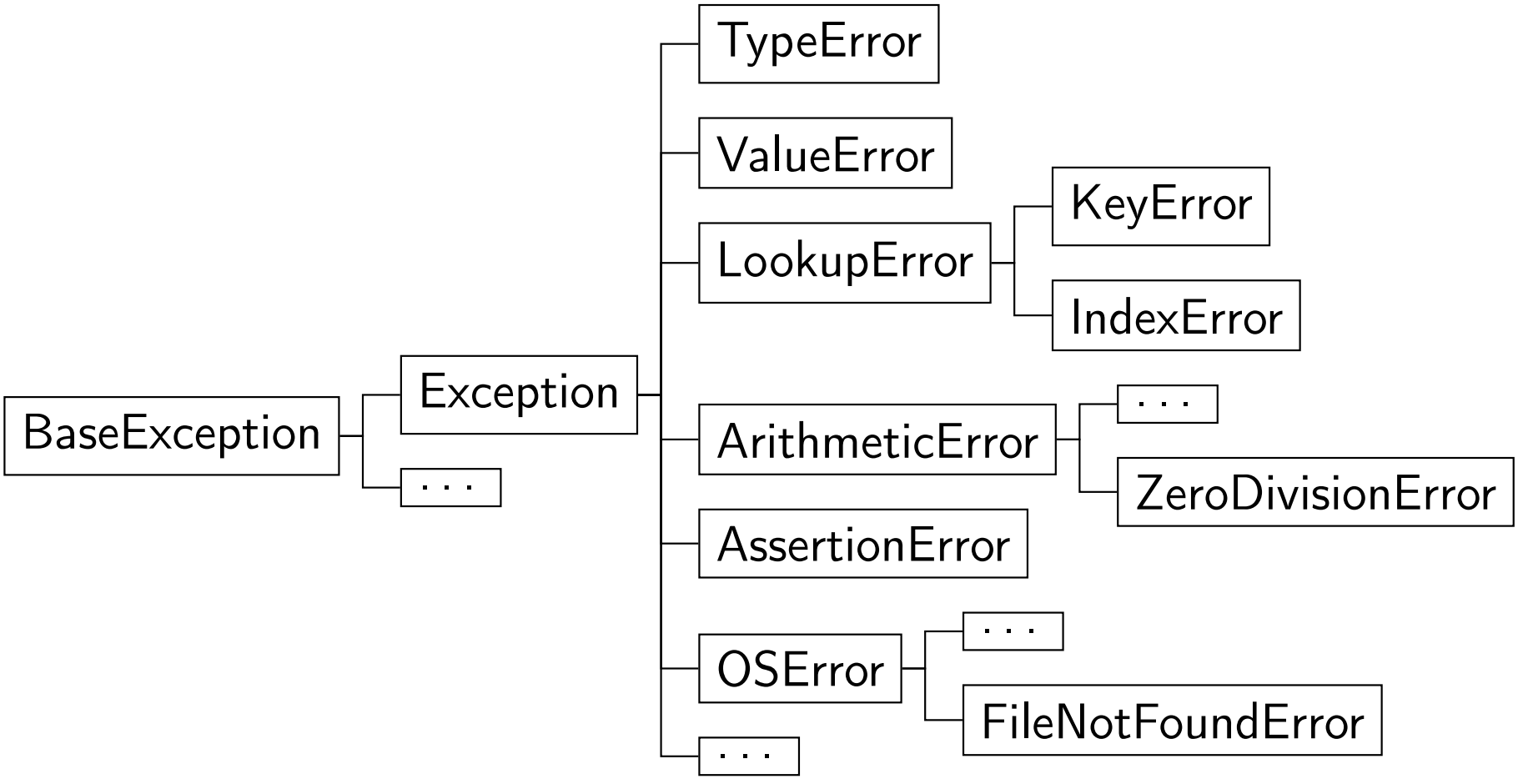
Exemple : voir suite

Exemple: try . . . except

Exemple :

```
try:
        x = 1.2
        a = int(x)
except ValueError:
        print("Erreur: type attendu 'int'")
except ZeroDivisionError:
        print("Erreur: Division par 0")
except:
        print("Erreur imprévue")
else:
        print("Pas d'erreur")
```

Principales Exceptions en Python



Liste complete :
https://docs.python.org/3/library/exceptions.html

raise Exception

- On peut décider qu'on a une exception si une expression ne respecte pas la logique métier (Les exigence de l'exercice).
- *raise* permet d'informer Python qu'on est en situation d'exception.
  Syntaxe: `raise <type d'exception>`
  Exemple

```
try:
        age = -1
        if age < 0:
                raise ValueError
except ValueError:
        print("Erreur: Valeur négative.")
else:
        print("Age correcte.")
```

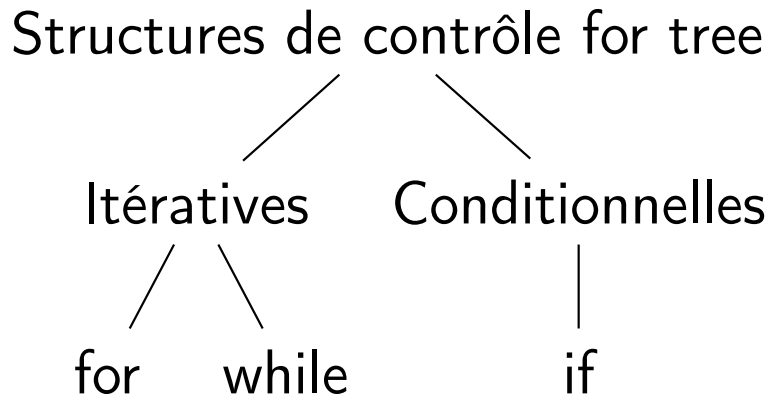
Utilisation de assert en Python

- L'instruction *assert* est utilisée pour tester si une condition est vraie. Si la condition est fausse, elle lève une exception `AssertionError`.
- `Cette méthode est utile pour vérifier les hypothèses pendant le développement.`

```
def division(a, b):
        assert b != 0, "Le diviseur ne doit pas être zéro"
        return a / b


try:
        result = division(10, 0)
except AssertionError as error:
        print(f"Erreur: {error}")
else:
        print(f"Résultat: {result}")
```

Structures de contrôle



Les structures de contrôle conditionnelles

Exemple

```
x = 10

if x > 0:
    print("x est positif")
elif x == 0:
    print("x est zéro")
else: # si tout est faux.
    print("x est négatif")
```

Les structures de contrôle conditionnelles

Exemple

```
x = 10

if x > 0:
    print("x est positif")
elif x == 0:
    print("x est zéro")
else: # si tout est faux.
    print("x est négatif")
```

- Opérateur ternaire :

```
r = True if x>0 else False
```

Les structures de contrôle conditionnelles

Exemple

```
x = 10

if x > 0:
    print("x est positif")
elif x == 0:
    print("x est zéro")
else: # si tout est faux.
    print("x est négatif")
```

- Opérateur ternaire :

```
r = True if x>0 else False
r = "+" if x>0 else "-" if x<0 else "0"
```

Les structures de contrôle itératives

Répeter l'exécution d'un bloc d'instrcutions

- `for i in range(start, end, step)`
  - Utilisée lorsque le nombre d'itérations est connu à l'avance.
  - Idéale pour parcourir des séquences comme des listes, des tuples, et des chaînes de caractères.
- Boucle `while condtition`
  - Utilisée lorsque le nombre d'itérations dépend d'une condition.
  - Pratique pour les situations où l'on doit attendre qu'une condition change.

Boucle `for` : Exemple

Exemple : Calcul de la somme des nombres impairs entre 1 à 10.

```
s = 0;  for i in range(1, 10, 2): s += i
```

La fonction **range()** génère une séquence de nombres, et la boucle **for** itère sur cette séquence.

```
range(start, stop, step)
```

- **start**: Valeur initiale (incluse), 0 par défaut.
- **stop**: Valeur finale (exclue), la séquence s'arrête avant ce nombre
- **step**: Incrémentation (optionnel), la différence entre chaque nombre dans la séquence. Si omis, le pas par défaut est 1.

Autrement

```
s = sum(range(1, 10, 2))
```

```
s = 0
for i in range(1,10):
    if i % 2 == 0: continue
    s += i
```

Boucle `while` : Exemple

Exemple : Calcul de la somme des nombres impairs entre 1 à 10.

```
s = i = 0
while i<10 :
    i+=1
    if i%2==1: s += i
    #else: continue
```

```
s = i = 0
while 2 : #True
    if i>10: break
    if i%2 : s += i
    i += 1
```

```
s,i = 0,10;
while i>0:
    s += i if i%2 else 0
    i -= 1
```

Table of contents

La fonction *print()*.

Opérations Entrée/Sortie

» `help(print)`

```
print(*args, sep=' ', end='\n', file=None, flush=False)
```

Description :

- `objects` : Les objets à afficher, convertis en chaînes de caractères comme le fait `str()`.
- `sep` : Chaîne de séparation entre les objets (par défaut, un espace " ").
- `end` : Chaîne ajoutée à la fin. Par défaut, un saut de ligne '`\n`'.
- `file` : Objet avec une méthode `write(string)` ; utilise `sys.stdout` par défaut.
- `flush` : Si `True`, force le vidage du flux.

La fonction *print()*.

Exemples

Exemple 1: Impression de plusieurs objets avec un séparateur personnalisé `print("Bonjour", "le monde", "!", sep="---")`

Exemple 2: Utilisation de ‘end‘ pour personnaliser la fin de ligne

```
print("Début de la ligne", end=" ")
print("fin de la même ligne")
```

Exemple 3: Impression dans un fichier

```
with open("exemple.txt", "w") as file:
    print("Bonjour le monde", file=file)
```

La fonction *print()*.

Exemples: Suite

Exemple 4: Impression avec conversion de type

```
num = 123
print("Le nombre est " + str(num))
```

Exemple 5: Impression avec formatage

```
name = "Alice"
age = 30
print("Nom : " + name + ", Âge : " + str(age))
print("Nom : {}, Âge : {}".format(name, age))
print(f"Nom : {name}, Âge : {age}")

print("Nom : %s, Âge : %d" % (name, age))
#%s :chaine, %d: entier, %f:float

print("Valeur de pi : {:.2f}".format(3.14159))
#.2f affiche le nombre avec 2 décimales: 3.14

print("Grand nombre : {:,}".format(1234567))
#:, ajoute des séparateurs de milliers: 1,234,567
```

La fonction *input()*.

Aide et Explication

- `input(prompt='')` est une fonction intégrée de Python, utilisée pour lire une chaîne de caractères depuis l'entrée standard (habituellement, le clavier).
- Lorsque la fonction `input()` est appelée, elle affiche le texte du `prompt` (message qui demande d'entrer des informations) à l'utilisateur, sans ajouter de saut de ligne à la fin.
- Note: Toutes les entrées sont lues comme des chaînes de caractères. Pour traiter d'autres types de données, il est nécessaire de convertir les chaînes avec des fonctions comme `int()` ou `float()`.

La fonction *input()*.

Exemples

Exemple 1: Lecture d'une entrée utilisateur

```
nom = input("Entrez votre nom: ")
print(f"Bonjour, {nom}!")
```

Exemple 2: Lecture et conversion de données numériques

```
age = input("Entrez votre âge: ")
print(type(age)) # str
try:
    age = int(age)  # Conversion de chaîne à entier
except:
    print("Erreur de valeur")
else:
    print(f"Vous avez {age} ans.")
```

La fonction *input()*.

Exemples: Suite

Exemple 3: Utilisation d'un prompt avec des indications

```
reponse = input("Voulez-vous continuer ? (oui/non): ")
if reponse.lower() == "oui":
    print("Vous avez choisi de continuer.")
else:
    print("Fin du programme.")
```

Table of contents

Résumé des Types de Fonctions en Python

- **Fonctions intégrées** : Fonctions prédéfinies disponibles par défaut, comme `print()` et `len()`.
- **Fonctions définies par l'utilisateur** : Fonctions créées par les développeurs pour réutiliser du code.
- **Fonctions anonymes (lambda)** : Fonctions sans nom utilisées pour des opérations simples.
- **Fonctions d'ordre supérieur** : Fonctions prenant d'autres fonctions en argument ou renvoyant des fonctions.
- **Fonctions récursives** : Fonctions qui s'appellent elles-mêmes pour résoudre des problèmes divisibles.

Fonctions intégrées

- Disponibles sans importation.
- Exemples :
  - `print()` : Affiche du texte à l'écran.
  - `len()` : Retourne la longueur d'une séquence.

Exemple

```
texte = "Bonjour"
print(len(texte))  # Affiche 7
```

Fonctions définies par l'utilisateur

- Créées pour organiser et réutiliser le code.
- Utilisent le mot-clé `def`.

Exemple

```
def addition(a, b):
    return a + b

print(addition(3, 5))  # Affiche 8
```

C'est quoi le mot-clé `return`?

- `return` termine l'exécution d'une fonction et renvoie une valeur à l'appelant.

Quand utiliser `return`?

- Utiliser `return` pour renvoyer le résultat d'une opération ou le contenu d'une fonction.
- Permet de récupérer des données traitées par la fonction.

Fonctions anonymes (lambda)

- Utilisées pour des opérations simples.
- Définies avec le mot-clé `lambda`.

Exemple

```
addition = lambda x, y: x + y
print(addition(3, 5))  # Affiche 8
```

Fonctions d'ordre supérieur

- Prennent d'autres fonctions en argument ou les renvoient.
- Exemples : `map()`, `filter()`.

Exemple

```
def appliquer_fonction(f, liste):
    return [f(x) for x in liste]

résultats = appliquer_fonction(lambda x: x * 2, [1, 2, 3])
print(résultats)  # Affiche [2, 4, 6]
```

Fonctions récursives

- S'appellent elles-mêmes pour résoudre des sous-problèmes.
- Utile pour des problèmes comme le calcul de factorielle.

Exemple

```
def factorielle(n):
    if n == 0:
        return 1
    else:
        return n * factorielle(n - 1)

print(factorielle(5))  # Affiche 120
```

Paramètres, Arguments, et Valeurs par Défaut

Paramètres:

- Ce sont les variables définies dans la signature d'une fonction.
- Ils représentent les valeurs que la fonction attend de recevoir lors de son appel.

Exemple:

```
def greet(name, age):
print(f"Bonjour {name}, vous avez {age} ans!")
```

Arguments:

- Ce sont les valeurs réelles passées à une fonction lors de son appel.

Exemple:

```
greet("Alice", 30)  # "Alice" et 30 sont des arguments
```

Paramètres, Arguments, et Valeurs par Défaut

Suite

Valeurs par défaut:

- Permettent de spécifier une valeur par défaut pour un paramètre si aucun argument n'est fourni lors de l'appel de la fonction.
- Utile pour simplifier les appels de fonction lorsque des valeurs courantes sont souvent utilisées.

Exemple:

```
def greet(name, age=25):
print(f"Bonjour {name}, vous avez {age} ans!")

greet("Bob") # age utilise la valeur par défaut de 25
greet("Carol", 40) # age est explicitement défini à 40
```

Les fonctions: Passage de paramètres

Passage par valeur

Passage par valeur:

- Les valeurs des variables sont copiées dans les paramètres de la fonction.
- Les modifications apportées aux paramètres dans la fonction n'affectent pas les variables d'origine.
- Exemple:

```
def increment(value):
    value += 1
    return value

x = 10
increment(x) # est différent de x = increment(x)
print(x)  # x est toujours 10
```

Les fonctions: Passage de paramètres

Passage par adresse

Passage par adresse

- Les variables passent des références aux objets, pas les objets eux-mêmes.
- Les modifications apportées à l'objet affectent l'objet d'origine.

Exemple:

```
def append_element(lst, elem):
    lst.append(elem)

my_list = [1, 2, 3]
append_element(my_list, 4)
print(my_list)  # my_list est maintenant [1, 2, 3, 4]
```

Variables locales et globales

Variables locales:

- Déclarées à l'intérieur d'une fonction.
- Valables uniquement dans le scope (portée) de cette fonction.
- Exemple:

```
def my_function():
    local_var = 5  # Variable locale
    print(local_var)

my_function()
print(local_var) #Erreur: local_var n'est pas définie ici
```

Variables locales et globales

Variables globales:

- Déclarées à l'extérieur de toute fonction.
- Valables dans tout le script, y compris dans les fonctions.

Exemple:

```
global_var = 10  # Variable globale
def my_function():
    print(global_var) # Peut accéder à la variable globale

my_function()
print(global_var) # Accessible ici aussi
```

Variables locales et globales

Suite

Utilisation de `global`

- Permet de modifier une variable globale à l'intérieur d'une fonction.

Exemple:

```
global_var = 10
def modify_global():
    global global_var
    global_var = 20

modify_global()
print(global_var)  # 20
```

Sans **global**, on aura deux variables différentes(locale et globale) de même nom mais avec des valeurs différentes.

Identificateurs (ID)

Fonction `id()` en Python:

- Retourne l'identifiant (adresse mémoire) d'un objet.
- Peut être utilisé pour vérifier si deux variables font référence au même objet en mémoire.

Exemple :

```
def verifier_id(a, b):
    print(f"Identifiant de a: {id(a)}")
    print(f"Identifiant de b: {id(b)}")
    if id(a) == id(b):
        print("a et b font référence au même objet.")
    else:
        print("a et b ne font pas référence au même objet.")
# Exemple d'utilisation
x = [1, 2, 3];    y = x ;    z = [1, 2, 3]
verifier_id(x, y) # a et b sont identiques, mêmes identifiants.
verifier_id(x, z) # a et b sont différents objets en mémoire,
                  # donc leurs identifiants seront différents.
```

Table of contents

Exercices

- Exercice 1

Table of contents

References

[Goldbach, 1742] Christian Goldbach.
A problem we should try to solve before the ISPN '43 deadline,
Letter to Leonhard Euler, 1742.

Open Questions

Is every even number the sum of two primes? [1]