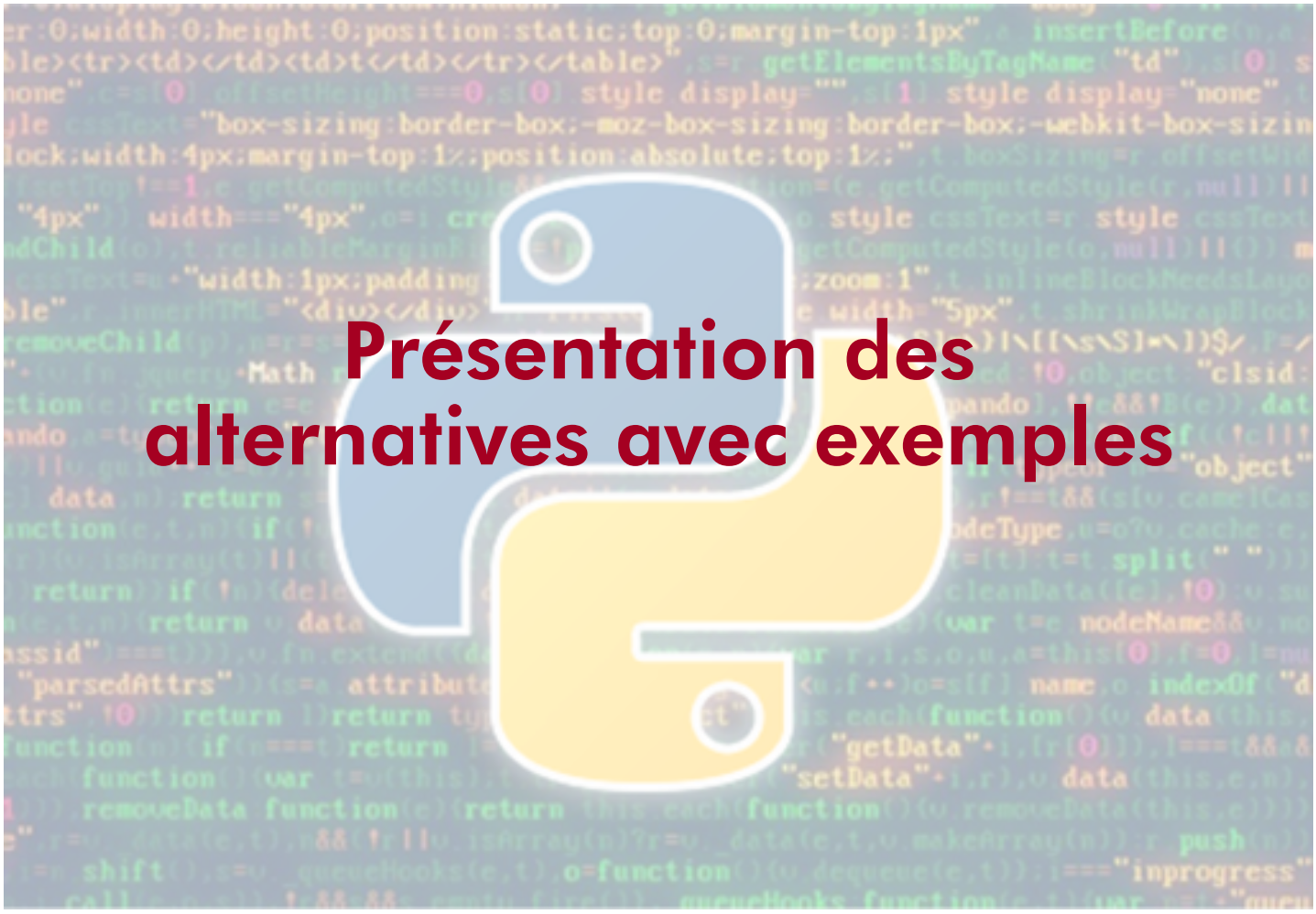
# #3 Introduction au langage Python

Les structures alternatives (ou conditionnelles)

*Édition 2021-2022*

## Plan



# Présentation des alternatives avec exemples

## Définition et différents types de structures conditionnelles (ou alternatives)

### Définition du vocabulaire relatif aux structures alternatives

- **Se sont les structures de contrôle qui permettent d'effectuer une ou plusieurs actions suivant la valeur de vérité attachée à une condition, c'est-à-dire la réponse, vraie ou fausse, à une question (qui est un prédicat).**

### Suivant le nombre de cas répondant à la condition, il y a différentes alternatives

- **Alternative simple (ou test simple)**
  - Pour une seule prise de décision : alternative « `if` »
  - Elle permet d'effectuer une ou plusieurs instructions si (si = *if*) la condition attachée est vérifiée.
- **Alternative complète (ou test complet)**
  - Pour deux prises de décision : alternative « `if … else` »
  - Elle permet d'effectuer une ou plusieurs instructions si une condition est vérifiée, et une ou plusieurs autres instructions dans le cas contraire (sinon = *else*).
- **Alternative de sélection (ou sélection)**
  - Pour un nombre variable de prises de décision : sélection « `if … elif … elif … else` »
  - Prolonge l'alternative complète en complétant la structure « `if … else` » avec des cas intermédiaires du type « sinon, si … » en cascade, ce qui contracte « `else-if` » en « `elif` »
  - Ici, dès que l'une des conditions intermédiaires est vérifiée, les instructions correspondantes sont effectuées, et le reste est ignoré. Dans ce type de structure, un seul bloc d'instructions **au plus** est exécuté.

### La condition (ou prédicat) prend la valeur vraie (`True`) ou la valeur fausse (`False`)

- `a = 2 ; a <= 0` **vaut** `False`
- `a = 2 ; b = -a/2 ; a % 2 == 0 and b < 0` **vaut** `True`

## Syntaxe complète des 3 alternatives : « if … », « if … else … » et «if … elif … else … »

### Alternative simple

```
if <condition>:                  # Entête de l'alternative, se termine par " :"
      <Début du bloc traité>     # si <condition> est vrai
      ...<Suite du bloc>...      # Le bloc est indenté…
      <Fin du bloc>              # sur plusieurs lignes.
<Instructions suivantes>         # rupture d'indentation = fin de bloc
```

### Alternative complète

```
if <condition>:            # Entête
      <Bloc traité>        # si <condition> est vraie
else:                      # else = sinon en anglais
      <Bloc traité>        # si <condition> est fausse
<Instructions suivantes> # rupture d'indentation = fin de bloc
```

***Remarques***

- Un seul « **if** » par alternative ;
- Un seul « **else** » au plus par alternative ;
- Autant de « **elif** » que l'on veut par alternative, mais pas forcément de « **else** » (il est facultatif)

### Sélection

```
if <condition1>:           # Entête
      <Bloc traité>        # si <condition1> est vraie
elif <condition2>:         # elif = else if = sélection,
    <Bloc traité>          # si <condition2> vraie (et <condition1> fausse)
elif ...:                  # succesion de elif
    <Bloc traité>          # si <condition2> vraie (et <condition1> fausse)
[else:                     # else = sinon en anglais [xx] signifie xx facultatif
      <Bloc traité>]       # si aucune des <conditionN> vraie
<Instructions suivantes>
```



## Alternative simple « if »

### Sur des exemples (rédigés dans la console)

**S'il n'y a qu'une seule instruction à traiter**

```
>>>if p % 2 == 0 : print('p est pair')
NameError: name 'p' is not defined
>>>p = 4 ; if p % 2 ==0 : print('p est pair')
p est pair # La réponse est « vraie »
>>>p = 37
>>>if p%2 == 0 : print('p est pair')
>>>
```

Les **…** indiquent que Python attend d'éventuelles nouvelles instructions. En validant (touche return), nous indiquons à l'interpréteur que le traitement peut commencer.

```
# Pas de réponse : elle est fausse, p impair
```

**S'il n'y a plusieurs instructions à traiter, elles forment un bloc qui doit être indenté**

```
>>>p = 36
>>>if p % 2 == 0 :
...print('p est pair')
.  File "<stdin>", line 2
.     print('p est pair')
Indentation Error : expected an indented block
>>>if p % 2 == 0 :
...    print('p est pair')
...    q = p // 2
...    print('p = 2 *', q)
...
p est pair
p = 2 * 18
```

L'absence d'indentation…
…provoque une erreur

Mais avec un 'tab' (⇥), c'est mieux
Le message énoncé en clair
La fin de l'indentation après `print` indique que le bloc d'instructions est terminé : nous quittons l'alternative.
Les deux affichages…
…en guise de résultat

## Alternative complète « if … else »

### Sur un exemple

```
>>>age = 21
>>>if age >= 18:
...     print('vous êtes majeur')
...else :
...     print('vous êtes mineur')
...
Vous êtes majeur
```

Dans ce cas un message correspondant à une action est affiché : il y aura donc toujours quelque chose d'affiché.
Il y a obligatoirement une indentation avant les instructions (actions) : c'est un bloc.

### Complément concernant les inégalités multiples

- **Il est possible d'écrire cette double inégalité**

```
>>>if 12 <= age <= 18:
...     print("C'est l'adolescence !")
```

C'est ce que l'on écrirait mathématiquement…

- **À la place de**

```
>>>if ag >= 12 and age <= 18:
...     print("C'est l'adolescence !")
```



## Alternative de sélection « if … elif … else »

### Sur un exemple

```
>>>rang = 501
>>>if rang <= 50:
...     print('Vous êtes grand admissible')
...elif rang <= 450:
...     print('Vous êtes sur liste principale')
...elif rang <= 550:
...     print('Vous êtes en liste complémentaire')
...else:
...     print('Vous êtes refusé')
...
Vous êtes sur liste complémentaire
```

Dans ce cas un message correspondant à une action est affiché : comme il y a toujours une option qui est choisie, il y aura toujours quelque chose d'affiché.
Il y a obligatoirement une indentation avant les instructions (actions) : c'est le bloc qui suit ':'.
Chaque bloc peut être composé d'une seule ou de plusieurs actions.

### Note(s) supplémentaire(s)

## Les opérateurs de comparaison et connecteurs logiques

### Déjà présentés dans le cours « 0 – Introduction au langage Python – généralités… », les voici rappelés

| Symbole des opérateurs | Signification en « clair » | Exemple |
|---|---|---|
| < | Strictement inférieur (infériorité stricte) | `'a'<'z'` (`True`) / `1000<999` (`False`) |
| > | Strictement supérieur (supériorité stricte) | `'A'>'B'` (`False`) / `1000<999` (`False`) |
| <= | inférieur ou égal (infériorité large) | `2<=2` (`True`) / `10000<=1000` (`False`) |
| >= | supérieur ou égal (supériorité large) | `2>=2` (`True`) / `10000>=1000` (`True`) |
| == | Égalité (identité) | `a=1 ; a==1` (`True`) / `'abc'=='ABC'` (`False`) |
| != | Différent de | `1!=2` (`True`) / `a=2 ; a!=2` renvoient `False` |

| Connecteurs logiques | Symboles | Exemples |
|---|---|---|
| identité logique (oui) | `is` | `'black' is 'white'` renvoie `False` |
| complément logique (non) | `not` ou `!` | `not (1!=2)` renvoie `False` |
| et (and) | `and` ou `&` | `a>b and c>d` renvoie `True` ou `False` |
| ou (or) | `or` ou \| (altGr+6 / ⇑+alt+L) | `a>=b or c>d` renvoie `True` ou `False` |
| ou exclusif (xor) | ^ (bit à bits) | Pas dans les connexions logiques |



## Des conditions, pas que dans les alternatives ; au travers de quelques exemples

### La condition est absolument nécessaires pour les boucles « `while` » (Cf. cours 4)

### Autres cas d'usage de conditions

### Expression conditionnelle (sur un exemple)

```
for v in range(10):
    res = 'pair' if v % 2 == 0 else
'impair'
    print('pour', v, ':', res)
```

L'expression commence par évaluer la condition après `if`. Si elle est vrai, alors `res = 'pair'` est évalué et sa valeur est renvoyée ; sinon, `'impair'` est affecté (si c'est une expression, elle est évaluée) et sa valeur est renvoyée.

### Les nombres ont aussi une valeur de vérité

```
for x in (-5, -2.1, 0, 2, 0.):
    if not x:
        print(x, '= 0, donc True')
    else:
        print(x, '!= 0, donc False')
```

La valeur 0 est assimilée à `False`, toutes les autres valeurs à `True`.

## Un exemple plus élaboré pour terminer

### Première solution : distinguer chaque cas avec une alternative de sélection

```
""" Programme testant si une année, saisie par l'utilisateur,
    est bissextile ou non """

annee = input("Saisissez une année : ") # Saisie de l'année testée
annee = int(annee) # Conversion chaîne -> nombre (ici un entier)
bissextile = False # initialisation booléenne de la réponse

if annee % 400 == 0:
    bissextile = True
elif annee % 100 == 0:
    bissextile = False
elif annee % 4 == 0:
    bissextile = True
else:
    bissextile = False

if bissextile: # Si l'année est bissextile
    print("L'année", annee, "est bissextile.")
else:
    print("L'année", annee, "n'est pas bissextile.")
```

**Vérification ici : https://fr.wikipedia.org/wiki/Année_bissextile**



## Optimisation de l'exemple

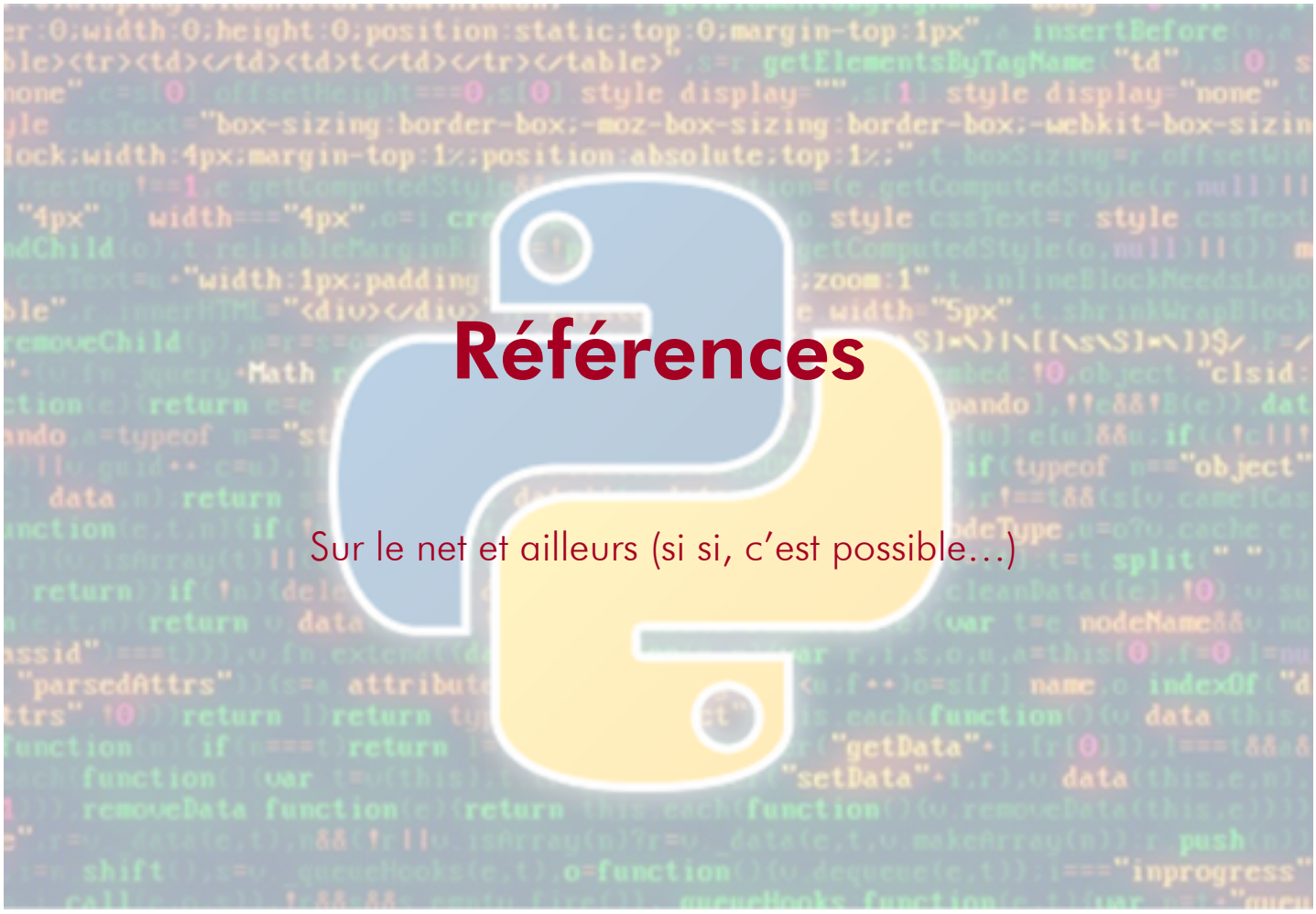
### Deuxième solution : regrouper les conditions morcelées

```
""" Programme testant si une année, saisie par l'utilisateur,
    est bissextile ou non """

# Programme testant si une année saisie par l'utilisateur
# est bissextile ou non

annee = input("Saisissez une année : ") # Saisie de l'année testée annee =
int(annee) # # Conversion chaîne -> nombre (ici un entier)

if annee % 400 == 0 or (annee % 4 == 0 and annee % 100 != 0):
    print("L'année", annee, "est bissextile.")
else:
    print("L'année", annee, "n'est pas bissextile.")
```



# Références

Sur le net et ailleurs (si si, c'est possible…)

## Références

### Livres

- Apprenez à programmer en Python, Vincent Le Goff, Openclassrooms, 2ème édition, 2014

### Wikipédia

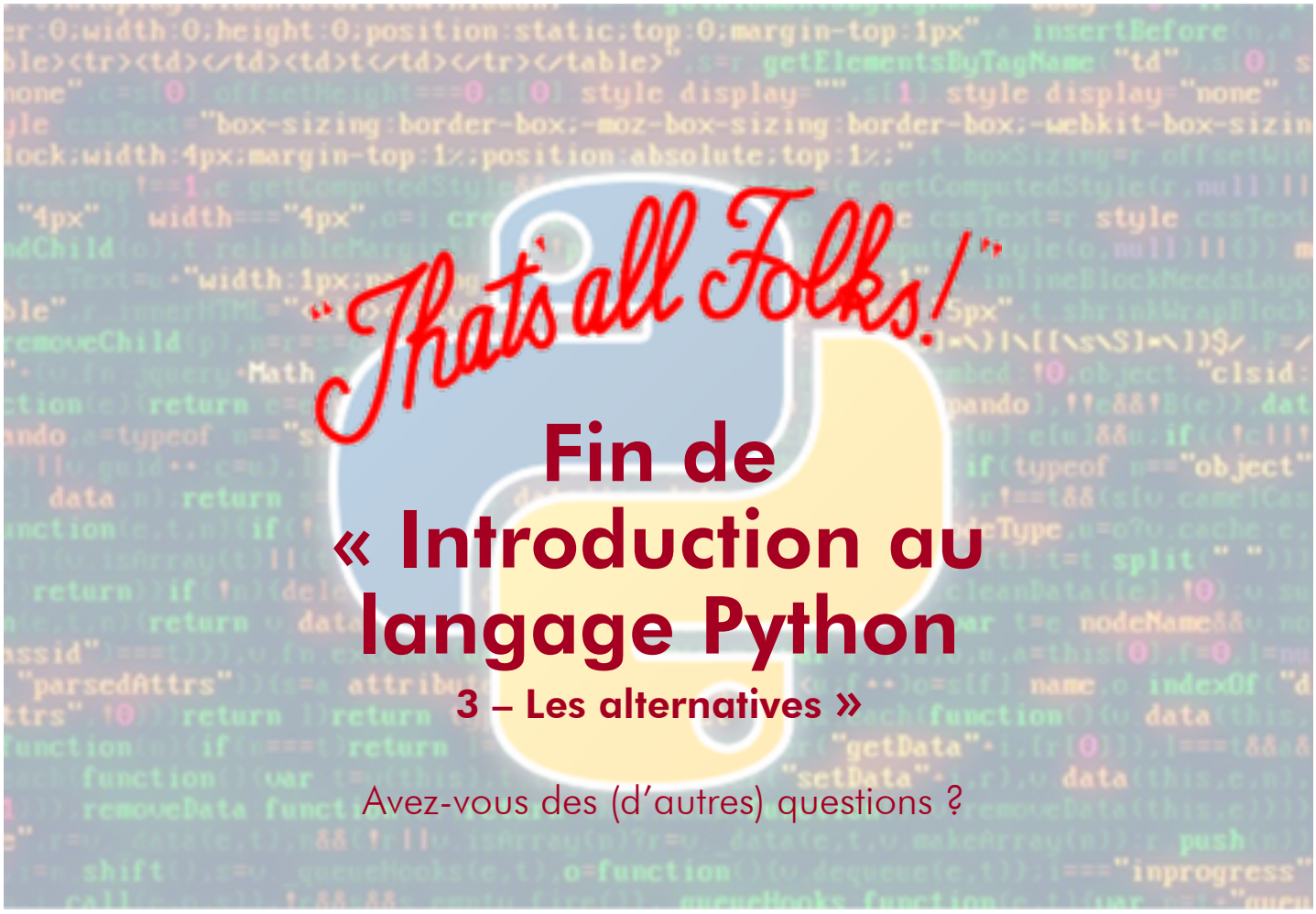
- https://fr.wikipedia.org/wiki/Python_(langage)

### Cours et références en ligne

- https://www.python.org
- https://www.developpez.com et https://python.developpez.com

### Autres cours et références

- Documents de cours d'informatique de Sophie Dion, lycée Châtelet, Douai
- Cours d'informatique de Xavier PESSOLES et Damien ICETA, UPSTI




# Fin de « Introduction au langage Python

## 3 – Les alternatives »

Avez-vous des (d'autres) questions ?