2.1 Structure des scripts Python

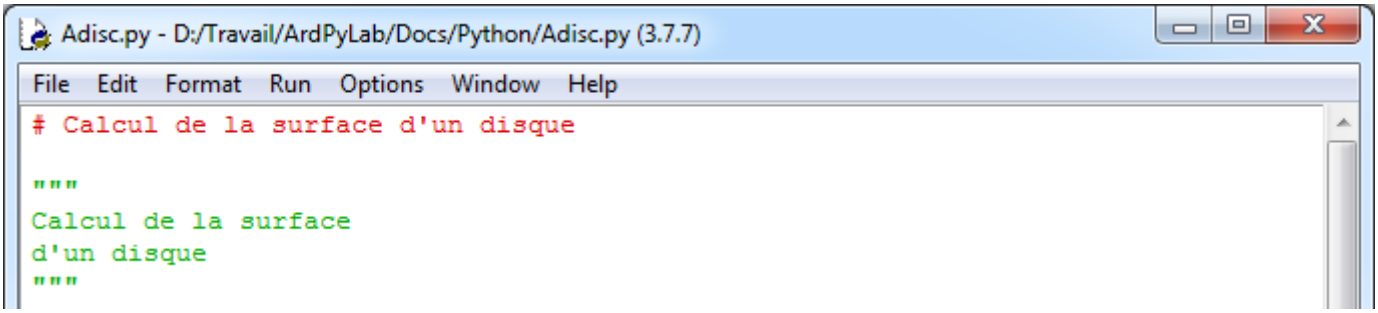
Un script Python est formée d'une suite d'instructions exécutées de haut en bas du script.

. Les instructions

Chaque instruction s'écrit sur une ligne, il n'y a pas de séparateur d'instruction. Si une ligne est trop grande, le caractère "****" permet de passer à la ligne suivante.

On utilise le caractère "**#** "pour insérer des commentaires dans un programme. Les commentaires vont du caractère "**#**" jusqu'à la fin de la ligne.

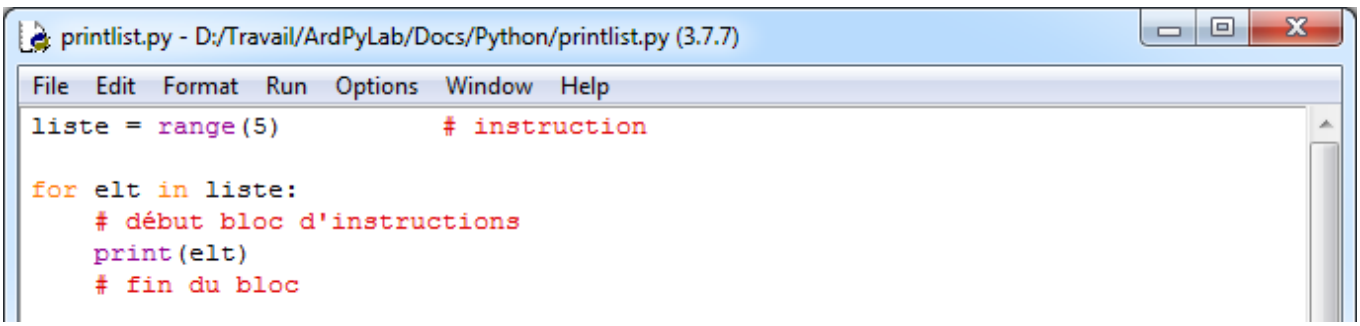
Il n'existe pas de commentaires en bloc comme en C **(/* ... */)**. Mais il est possible d'utiliser des triples guillemets doubles ou simples (**'''**) avant et après le bloc de commentaires comme pour déclarer une variable chaine de caractères sur plusieurs lignes. Le bloc n'étant pas rattaché à une variable, il sera ignoré par l'interpréteur.
Sous IDLE, le plus simple est cependant de sélectionner le bloc de commentaires et de le déclarer comme tel avec "**Format/comment out region**".



. Les blocs d'instructions

Les blocs d'instructions sont matérialisés par des indentations (plus de **{** et **}** comme en C !).

Le caractère "**:**" sert à introduire les blocs.



. Les entrées-sorties

L'utilisateur a généralement besoin d'interagir avec le programme. En l'absence d'interface graphique, en mode "console" (Python shell), on doit pouvoir saisir ou entrer des informations, ce qui est fait depuis une lecture au clavier. Inversement, on doit pouvoir afficher ou sortir des informations, ce qui correspond généralement à une écriture sur l'écran (dans la console).

- Les entrées

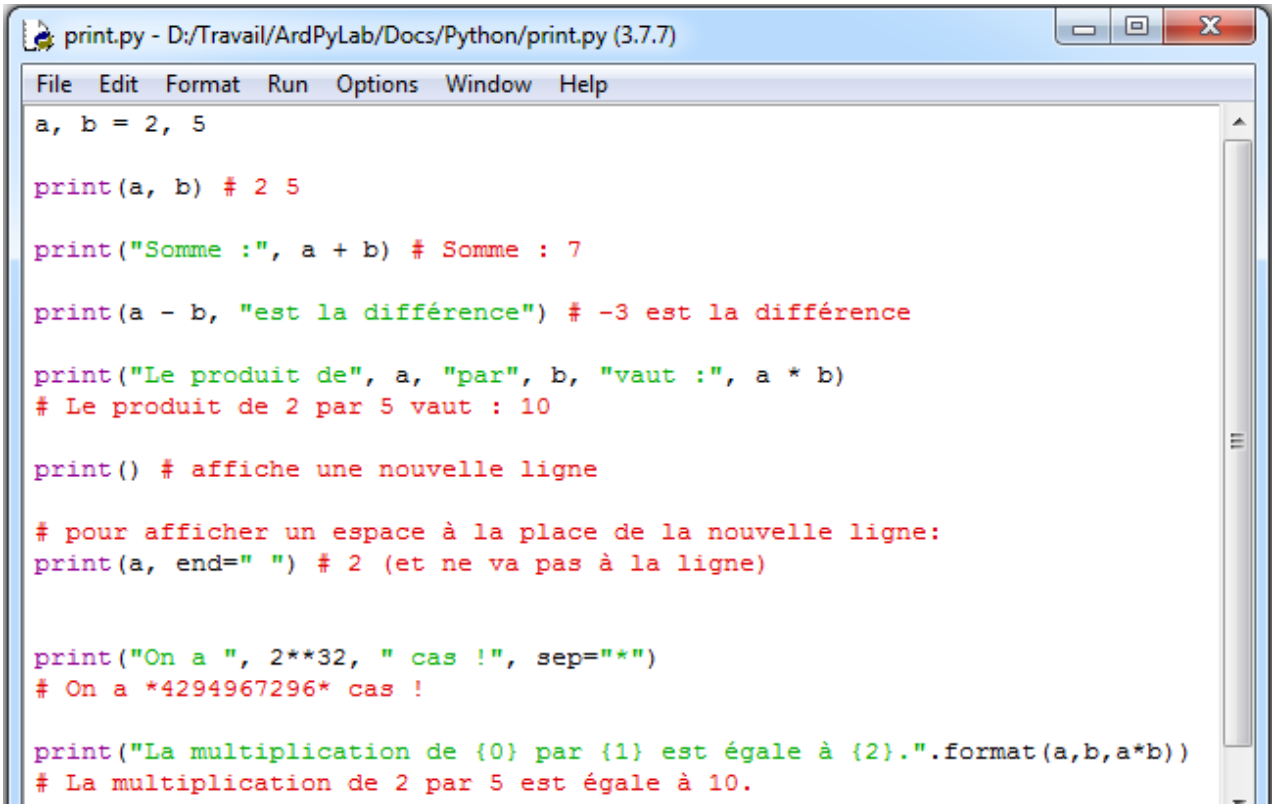
Il s'agit de réaliser une saisie dans la console Python. Pour cela, on utilise la fonction "**input()**" qui interrompt le programme, affiche une éventuelle invite et attend que l'utilisateur entre une donnée et la valide par la touche "**Entrée** " .

La fonction "**input()**" effectue toujours une saisie en mode texte (la saisie est une chaine – type '**str**') dont on peut ensuite changer le type par une conversion.

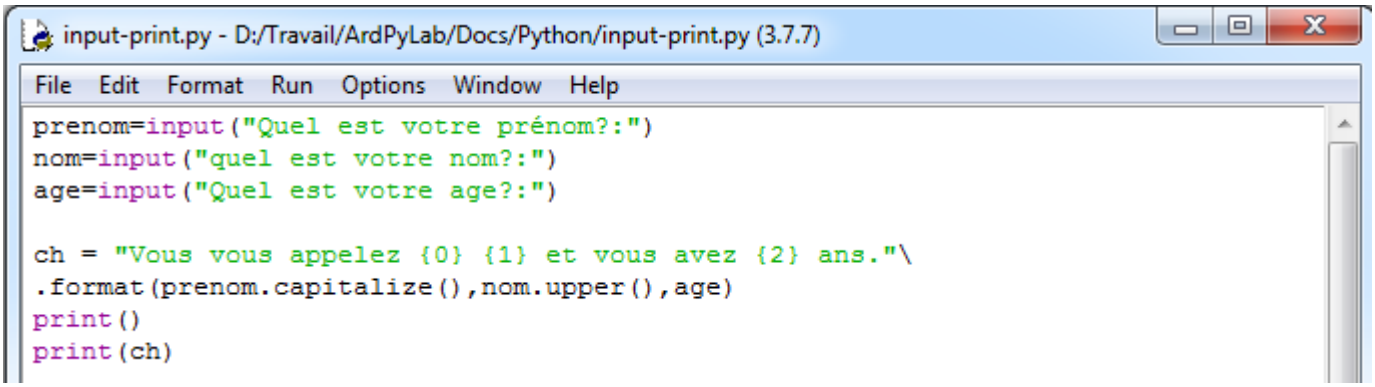
```
>>> nb = input("veuillez saisir un nombre:")
veuillez saisir un nombre:45
>>> print(type(nb))
<class 'str'>
>>> ch = input("veuillez saisir une chaine de caractères:")
veuillez saisir une chaine de caractères:azert123
>>> print(type(ch))
<class 'str'>
```

- Les sorties

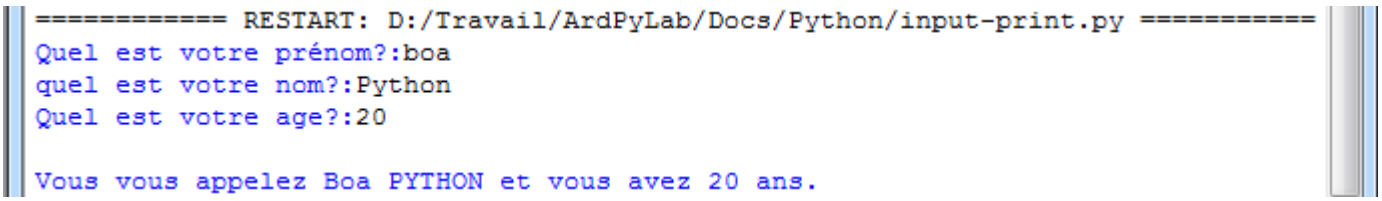
En mode interactif, Python lit, évalue et affiche, mais la fonction "**print()**" reste indispensable aux affichages dans les scripts :



Exemple : Le programme ci-dessous demande le nom, le prénom et l'age de l'utilisateur et affiche les données après formatage.



Résultat dans la fenêtre Python shell :



. Les conversions

Il existe un certain nombre de fonctions permettant de convertir les données d'un type à l'autre.
La fonction "**type()**" permet de récupérer le type de la donnée sous forme d'une chaîne.

Fonction	Description	Exemple
ord	retourne la valeur ASCII d'un caractère	ord('A')
chr	retourne le caractère à partir de sa valeur ASCII	char(65)
str	convertit en chaîne	str(10), str([10,20])
int	interprète la chaîne en entier	int('45')
int long	interprète la chaîne en entier long	long('56857657695476')
float	interprète la chaîne en flottant	float('23.56')

Autre conversions :

. **Conversion binaire**

La fonction "**bin()**" permet de convertir un nombre binaire en chaine de caractères :

```
>>> bin(204)
'0b11001100'
```

Pour convertir une chaine de caractères représentant un nombre binaire (base 2), on utilise la fonction **int()** en précisant la base :

```
>>> int('11001100',2)
204
```

Le préfixe 0b n'est pas obligatoire et peut être supprimé.

. **Conversion hexadécimale**

La fonction "**hex()**" permet de convertir un nombre hexadécimal en chaine de caractères et la fonction **int()** en précisant la base 16 fait la conversion inverse:

```
>>> hex(32)
'0x20'
>>> int('0x20',16)
32
```

. Les structures de contrôles

- Condition **if** :

L'instruction **if** ("si" en français), utilisée avec un opérateur logique de comparaison, permet de tester si une condition est vraie.

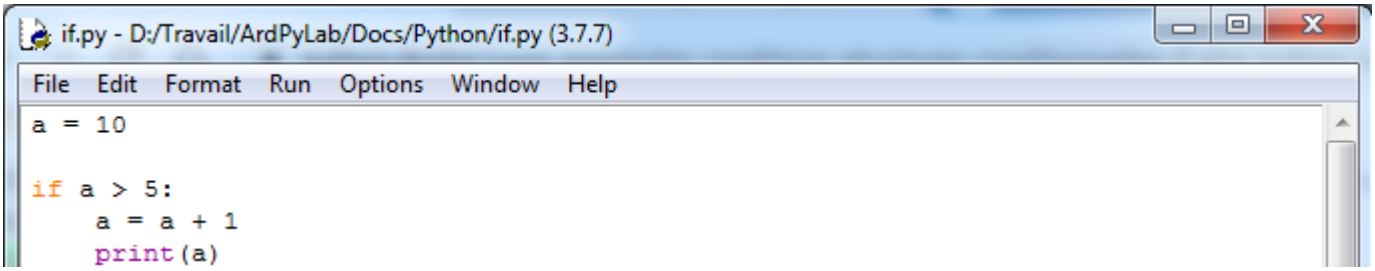
Le format d'un test **if** est le suivant :

```
if  uneVariable > 50 :
    # Instructions  (attention aux tabulations !)
```

Dans cet exemple, le programme va tester si la variable **uneVariable** est supérieure à 50. Si c'est le cas, le programme va réaliser une action particulière. Autrement dit, si l'état du test est vrai, le bloc d'instructions (ligne d'instructions de même indentation) après le caractère "**:**" est exécuté.

Exemple :

Une valeur est donnée à une variable et si cette valeur est supérieure à 5, alors celle-ci est incrémentée de 1 et affichée.



Résultat dans la fenêtre Python Shell :

```
================ RESTART: D:/Travail/ArdPyLab/Docs/Python/if.py ================
11
>>>
```

Rappel des opérateurs logiques de comparaison :

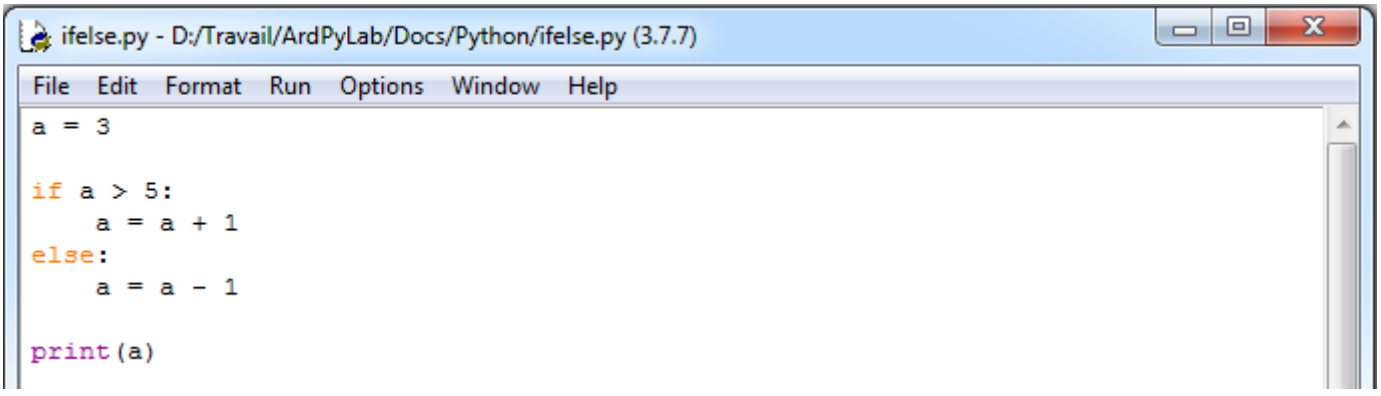
- x == y est vrai quand x est égal à y,
- x != y est vrai quand x est différent de y,
- x > y est vrai quand x est strictement supérieur à y,
- x < y est vrai quand x est strictement inférieur à y,
- x >= y est vrai quand x est supérieur ou égal à y,
- x <= y est vrai quand x est inférieur ou égal à y.

- Condition **if / else** :

L'instruction **if / else** (si/sinon en français) permet un meilleur contrôle du déroulement du programme que la simple instruction if, en permettant de grouper plusieurs tests ensemble.

```
if  var > 10:
    #action A
else:
    #action B
```

Exemple :



Résultat dans la fenêtre Python Shell :

```
===== RESTART: C:\Users\Olivier\Docs\Travail\ArdPyLab\Docs\Python\ifelse.py ====
2
>>>
```

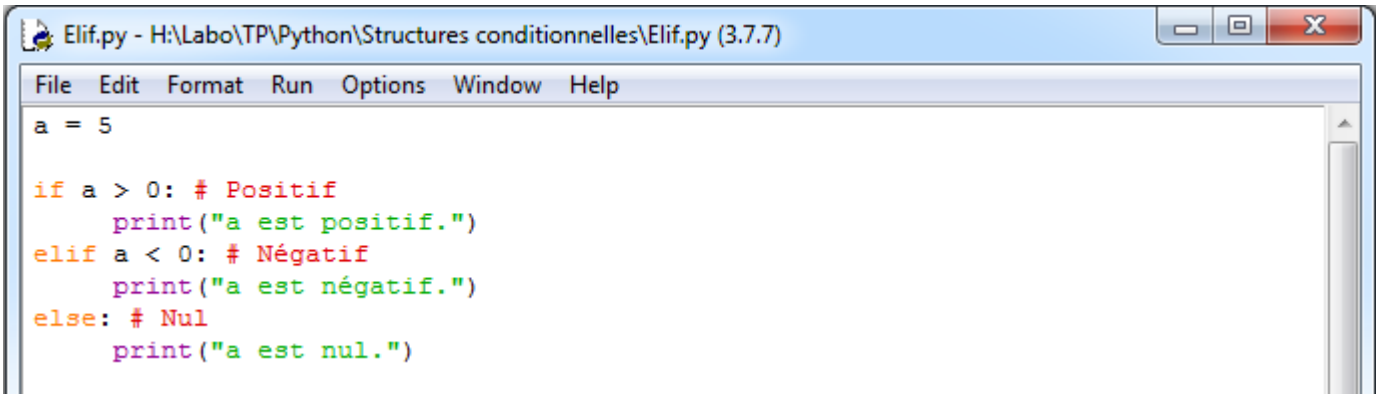
- Condition **elif** :

L'instruction **else** peut contenir un autre test **if**, et donc des tests multiples, mutuellement exclusifs peuvent être réalisés en même temps. On peut alors utiliser le mot clé **elif** à la place de **else : if ...**.

Chaque test sera réalisé après le suivant jusqu'à ce qu'un test VRAI soit rencontré.
Quand une condition vraie est rencontrée, les instructions associées sont réalisées, puis le programme continue son exécution à la ligne suivant l'ensemble de la construction **if/elif**.
Si aucun test n'est VRAI, le bloc d'instructions par défaut **else** est exécuté, s'il est présent, déterminant ainsi le comportement par défaut.

Un bloc **elif** peut être utilisé avec ou sans bloc de conclusion **else**.
Un nombre illimité de branches **elif** est autorisé.

Exemple :



- **AND / OR**

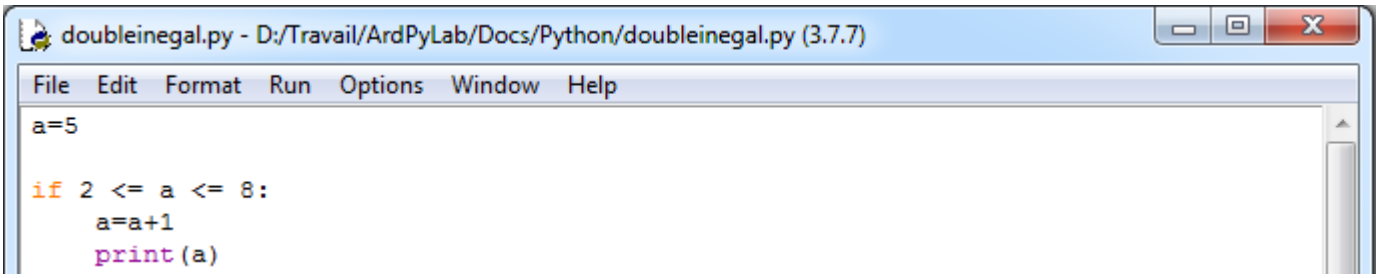
Il est possible d'affiner une condition avec les mots clé **AND** qui signifie " ET " et **OR** qui signifie " OU ".
Ces opérateurs peuvent être utilisés à l'intérieur de la condition d'une instruction if pour associer plusieurs conditions à tester.

```
if var >= 5  and var <= 10 :  # est VRAI seulement si var appartient à l'intervalle [5;10]
      # bloc d'instructions
else:
      # bloc d'instructions
```

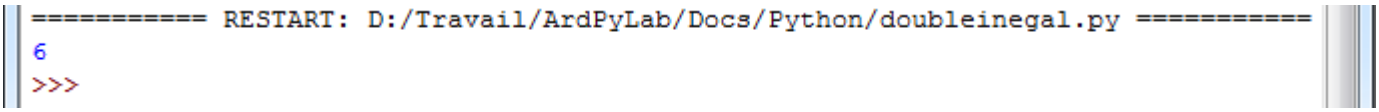
```
if var1 > 0 or var2 > 0 : # est vrai si var1 supérieur à 0 ou si var2 supérieur à 0
    # bloc d'instructions
else:
    # bloc d'instructions
```

Remarque :

Python permet aussi l'enchaînement des comparaisons à l'aide d'une double inégalité.



Résultat dans la fenêtre Python Shell :



- les structures itératives :

. boucles **While**

Les boucles while ("tant que" en anglais) bouclent sans fin, et indéfiniment, jusqu'à ce que la condition ou l'expression testée devienne fausse.
Quelque chose doit modifier la variable testée, sinon la boucle while ne se terminera jamais. C'est généralement une variable incrémentée dans le bloc d'instructions de la boucle.

```
var = 0
while var < 10 :  # tant que la variable est inférieur à 10
    # fait quelque chose 10 fois de suite...
    var += 1 # incrémente la variable
```

Exemple : Affichage d'une table de multiplication

while.py - H:\Labo\TP\Python\Boucles\while.py (3.7.7)

File Edit Format Run Options Window Help

```
nb = 7
i = 0

while i < 10: # Tant que i est strictement inférieure à 10
    print(i + 1, "*", nb, "=", (i + 1) * nb)
    i += 1 # On incrémente i de 1 à chaque tour de boucle
```

Résultat dans la fenêtre Python Shell :

```
================== RESTART: H:\Labo\TP\Python\Boucles\while.py ==================
1 * 7 = 7
2 * 7 = 14
3 * 7 = 21
4 * 7 = 28
5 * 7 = 35
6 * 7 = 42
7 * 7 = 49
8 * 7 = 56
9 * 7 = 63
10 * 7 = 70
```

Remarques :

Il existe des mots clés qui permettent d'effectuer une rupture dans la boucle itérative :

. **continue** (continue directement à la prochaine itération de la boucle, la boucle est court-circuitée)

Exemple :

Dans l'exemple suivant, tant que la variable i est inférieure à 20, celle-ci est incrémentée de 1 jusqu'à ce qu'elle soit égale à un multiple de 3. Elle est alors incrémentée de 4 et la boucle normale est reprise à l'aide de l'instruction **continue**.

while2.py - H:\Labo\TP\Python\Boucles\while2.py (3.7.7)

File Edit Format Run Options Window Help

```
i = 1
while i < 20:
    if i % 3 == 0:
        i += 4 # On ajoute 4 à i
        print("On incrémente i de 4. i est maintenant égale à", i)
        continue
    print("La variable i =", i)
    i += 1 # Dans le cas classique on ajoute juste 1 à i
```

Résultat dans la fenêtre Python Shell :

```
================ RESTART: H:\Labo\TP\Python\Boucles\while2.py ================
La variable i = 1
La variable i = 2
On incrémente i de 4. i est maintenant égale à 7
La variable i = 7
La variable i = 8
On incrémente i de 4. i est maintenant égale à 13
La variable i = 13
La variable i = 14
On incrémente i de 4. i est maintenant égale à 19
La variable i = 19
```

. **break** (sort de la boucle en cours)

. **pass** (instruction vide – ne fait rien)

Il est parfois pratique d'utiliser une boucle **while** infinie (dont la condition est toujours vraie), et d'utiliser les ruptures de séquences.

```
while True:
    # bloc d'instructions
    if condition : break
```

Exemple :

Dans cet exemple, on demande à l'utilisateur de saisir la lettre 'Q' pour sortir de la boucle.

Break.py - H:\Labo\TP\Python\Boucles\Break.py (3.7.7)

File Edit Format Run Options Window Help

```
while True: # True est toujours vrai -> boucle infinie
    lettre = input("Tapez 'Q' pour quitter : ")
    if lettre == "Q":
        print("Fin de la boucle")
        break
```

. boucles **for**

L'utilisation principale de l'instruction **for** est de parcourir un itérable, c'est-à-dire un conteneur que l'on peut parcourir élément par élément, dans l'ordre ou non, suivant son type :

- Parcours d'une chaine de caractères :

for.py - D:/Travail/ArdPyLab/Docs/Python/for.py (3.7.7)

File Edit Format Run Options Window Help

```
s = "azerty"
for lettre in s:
    print(lettre)
```

Résultat dans la fenêtre Python Shell :

```
========== RESTART: D:/Travail/ArdPyLab/Docs/Python/for.py ==========
a
z
e
r
t
y
```

- Parcours d'une liste :

for2.py - D:/Travail/ArdPyLab/Docs/Python/for2.py (3.7.7)

File Edit Format Run Options Window Help

```
liste = ["A",2,3,"azerty"]
for elt in liste:
    print(elt)
```

Résultat dans la fenêtre Python Shell :

```
========== RESTART: D:/Travail/ArdPyLab/Docs/Python/for2.py ==========
A
2
3
azerty
```

L'instruction **for** peut également être est utilisée pour répéter l'exécution d'un bloc d'instructions à l'aide de la fonction **range()** déjà vu lors de la présentation des listes :

```
for i in range(10):
    # bloc d'instructions
```

Dans cet exemple, le bloc d'instructions sera exécuté 10 fois.

Remarques:

- Une boucle **for** peut également être arrêtée avec l'instruction **break**

Exemple :

Dans cet exemple, la boucle **for** parcours les éléments d'une liste de nombres. Si le nombre lu est supérieur à 15, la boucle est stoppée.

for3.py - D:/Travail/ArdPyLab/Docs/Python/for3.py (3.7.7)

File Edit Format Run Options Window Help

```
liste = [1,5,10,15,20,25]
for i in liste:
    if i > 15:
        print("On stoppe la boucle")
        break
    print(i)
```

Résultat dans la fenêtre Python Shell :

```
============== RESTART: D:/Travail/ArdPyLab/Docs/Python/for3.py ==============
1
5
10
15
On stoppe la boucle
>>>
```

- Les boucles **while** et **for** peuvent posséder une clause **else** qui ne s'exécute que si la boucle se termine normalement, c'est-à-dire sans interruption avec l'instruction **break** :

while_else.py - D:/Travail/ArdPyLab/Docs/Python/while_else.py (3.7.7)

File Edit Format Run Options Window Help

```
liste = ['A', 'B', 'C', 'D']
element = 'B'

i = 0
while i < len(liste):
    if liste[i] == element:
        # element trouvé
        print(element, " est dans la liste, à l'indice: ",i)
        break
    i += 1
else:
    # element non trouvé
    print(element, " n'est pas dans la liste.")
```

Ln: 13 Col: 47

Dans l'exemple ci-dessus, on parcourt une liste à l'aide d'une boucle **while** pour savoir si la variable **element** est dans la liste. Si la variable est trouvée, la boucle est stoppée avec une instruction break :

Résultat dans la fenêtre Python Shell :

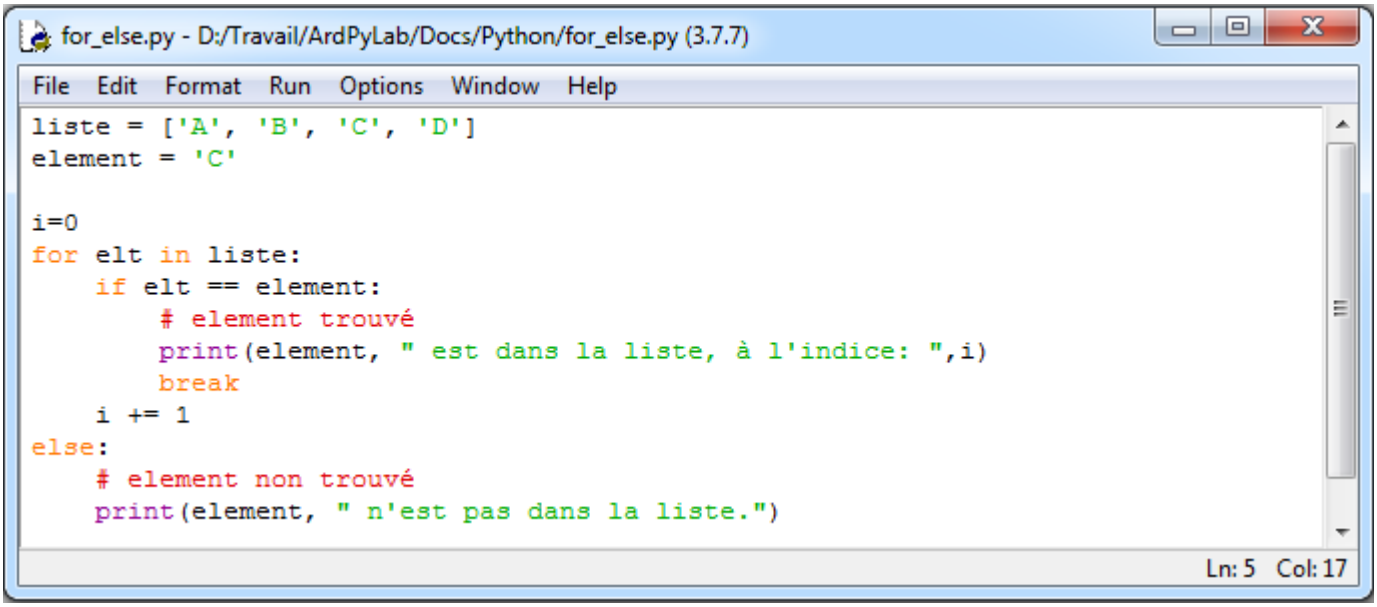
```
============ RESTART: D:/Travail/ArdPyLab/Docs/Python/while_else.py ============
B  est dans la liste, à l'indice:  1
>>>
```

, sinon on affiche que la variable n'est pas dans la liste :

Résultat dans la fenêtre Python Shell avec modification de la variable **element = 'E'** :

```
========== RESTART: D:/Travail/ArdPyLab/Docs/Python/while_else.py ==========
E  n'est pas dans la liste.
>>>
```

Il en est de même avec une boucle **for** :



Résultats dans la fenêtre Python Shell :

```
============= RESTART: D:/Travail/ArdPyLab/Docs/Python/for_else.py =============
C  est dans la liste, à l'indice:  2
>>>
```

avec modification de la variable **element = 'E'** :

```
============= RESTART: D:/Travail/ArdPyLab/Docs/Python/for_else.py =============
E  n'est pas dans la liste.
>>>
```

. Les exceptions – Gestion des erreurs dans les scripts

Lorsqu'une instruction d'un script ne se déroule pas correctement (par exemple, une division par zéro), une **exception est levée** ce qui interrompt le contexte d'exécution, pour revenir à un environnement d'exécution supérieur, jusqu'à celui gérant cette exception.

Par défaut, l'environnement supérieur est le shell de commande depuis lequel l'interpréteur Python a été lancé, et le comportement de gestion par défaut est d'afficher l'exception :

```
============= RESTART: D:/Travail/ArdPyLab/Docs/Python/exception.py ============
Traceback (most recent call last):
  File "D:/Travail/ArdPyLab/Docs/Python/exception.py", line 2, in <module>
    print(a/b)
ZeroDivisionError: division by zero
```

Pour gérer l'exception, et éviter la fin du programme, il faut utiliser la structure **try** et **except** :

```
try:
        # bloc d'instructions susceptibles d'échouer
except:
        # bloc d'instructions à faire en cas d'échec
```

Toutes les exceptions levées par Python sont des instances de sous-classe de la classe **Exception**.

La hiérarchie des sous-classes offre plusieurs exceptions standard, comme **ValueError** (exception levée quand on tente par exemple de convertir en nombre une chaine de caractères ne représentant pas un nombre) ou **ZeroDivisionError** (quand on tente de diviser un nombre par zéro).

Il est possible de compléter la structure **try except** avec un bloc **else** et un bloc **finally**. Les instructions du bloc **else** ne sont exécutées qu'en l'absence d'erreur et les instructions du bloc **finally** sont toujours effectuées quel que soient les erreurs rencontrées lors de l'exécution du bloc **try** ou en l'absence d'erreur.

La syntaxe complète d'une exception est alors :

```
try:
        ... # séquence normale d'exécution
except exception_1:
```

```
        ... # traitement de l'exception 1
except exception_2:
        ... # traitement de l'exception 2
else:
        ... # bloc d'instructions exécutées en l'absence d'erreur
finally:
        ... # bloc d'instructions toujours exécutées
```

Exemple :

Ce programme demande à l'utilisateur de saisir un nombre et tente de le convertir en flottant et de faire une division avec ce nombre :

exception3.py - D:/Travail/ArdPyLab/Docs/Python/exception3.py (3.7.7)

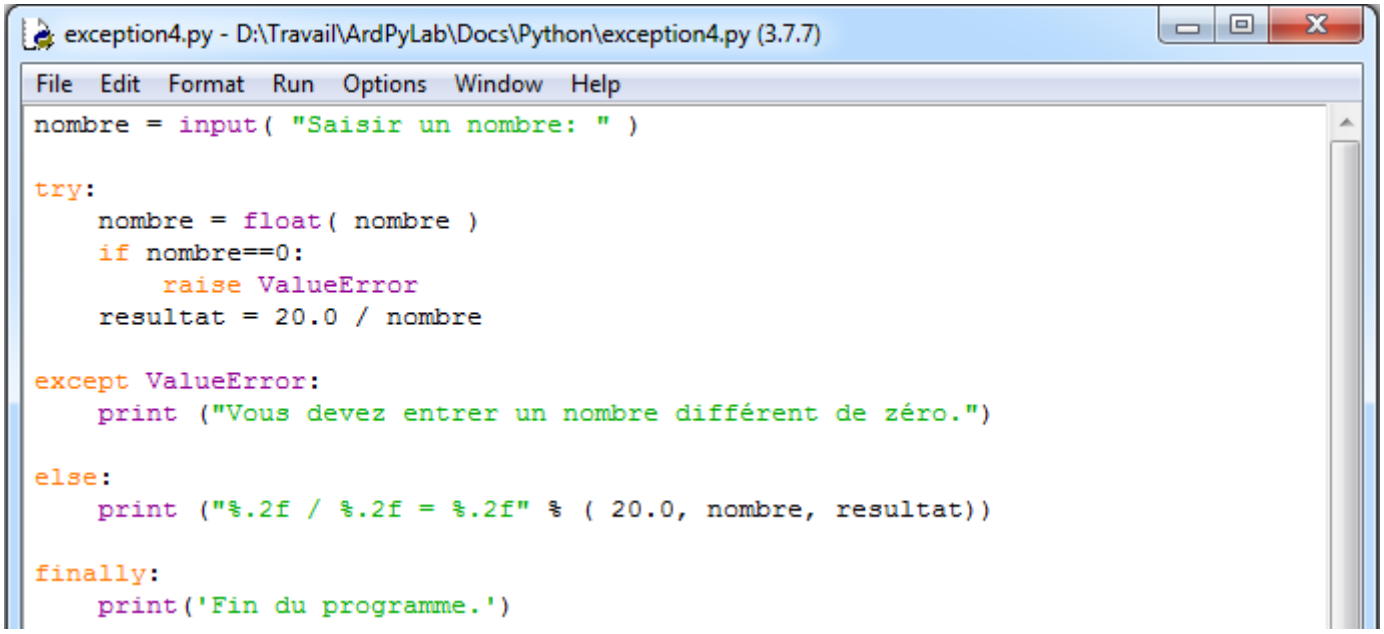
File Edit Format Run Options Window Help

```
nombre = input( "Saisir un nombre: " )

try:
    nombre = float( nombre )
    resultat = 20.0 / nombre

except ValueError:
    print ("Vous devez entrer un nombre.")
except ZeroDivisionError:
    print ("Essai de division par zéro.")

else:
    print ("%.2f / %.2f = %.2f" % ( 20.0, nombre, resultat))

finally:
    print('Fin du programme.')
```

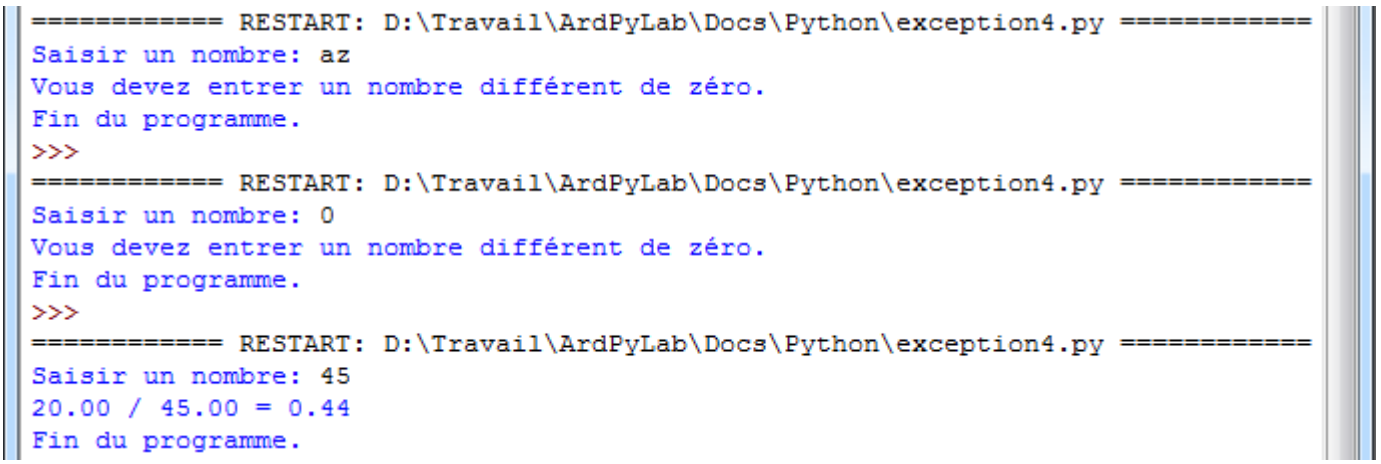
Résultats dans la fenêtre Python Shell :

```
============ RESTART: D:/Travail/ArdPyLab/Docs/Python/exception3.py ============
Saisir un nombre: za
Vous devez entrer un nombre.
Fin du programme.
>>>
============ RESTART: D:/Travail/ArdPyLab/Docs/Python/exception3.py ============
Saisir un nombre: 0
Essai de division par zéro.
Fin du programme.
>>>
============ RESTART: D:/Travail/ArdPyLab/Docs/Python/exception3.py ============
Saisir un nombre: 12
20.00 / 12.00 = 1.67
Fin du programme.
```

L'instruction **raise** permet de lever volontairement une exception. Ainsi le script du programme précédent devient :



Après la conversion de la chaine saisie au clavier en nombre, un test est effectué sur le nombre, si celui-ci est égale à 0, l'exception **ValueError** est levée à l'aide de l'instruction **raise**. Et bien-sûr si la chaine ne peut pas être convertie l'exception **ValueError** est également levée (c'est aussi le bloc d'exception du bloc **try**) :



Il est également possible de lever une exception avec l'instruction **assert**. Cette instruction va tester la condition mise juste après, et si elle est fausse, va lever une exception de type **AssertionError,** ce qui donne pour notre exemple :

exception5.py - D:\Travail\ArdPyLab\Docs\Python\exception5.py (3.7.7)

File Edit Format Run Options Window Help

```
nombre = input( "Saisir un nombre: " )

try:
    nombre = float( nombre )
    assert nombre > 0
    resultat = 20.0 / nombre

except AssertionError:
        print("Le nombre saisi doit être supérieur à 0.")

except ValueError:
    print ("Vous devez entrer un nombre.")

else:
    print ("%.2f / %.2f = %.2f" % ( 20.0, nombre, resultat))

finally:
    print('Fin du programme.')
```

Résultats dans la fenêtre Python Shell :

```
========== RESTART: D:\Travail\ArdPyLab\Docs\Python\exception5.py ==========
Saisir un nombre: az
Vous devez entrer un nombre.
Fin du programme.
>>>
========== RESTART: D:\Travail\ArdPyLab\Docs\Python\exception5.py ==========
Saisir un nombre: 0
Le nombre saisi doit être supérieur à 0.
Fin du programme.
>>>
========== RESTART: D:\Travail\ArdPyLab\Docs\Python\exception5.py ==========
Saisir un nombre: 45
20.00 / 45.00 = 0.44
Fin du programme.
```

La structure **Try … Except** est donc très utile pour tout ce qui est vérification des saisies au clavier. Mais contrairement aux scripts précédents, il est préférable que le programme ne s'arrête pas si la saisie au clavier ne satisfait pas au programme.

On utilisera pour cela une boucle **while**, afin de redemander à l'utilisateur une saisie au clavier si la précédente n'est pas adéquate, comme dans l'exemple suivant :

Dans ce programme, on demande à l'utilisateur de saisir un nombre supérieur à 0 et on vérifie si c'est un nombre premier.

exception2.py - D:\Travail\ArdPyLab\Docs\Python\exception2.py (3.7.7)

File Edit Format Run Options Window Help

```
nombresaisi = False

while nombresaisi == False:

    nombre = input("Saisissez un nombre : ")

    try:
        nombresaisi = True
        nombre = int(nombre)
        assert nombre > 0

        i = 2
        while i < nombre and nombre % i != 0:
            i = i + 1

        if i == nombre:
            print("Le nombre", nombre, "est premier.")
        else:
            print("Ce n'est pas un nombre premier.")

        break

    except AssertionError:
        print("Le nombre saisi est inférieur ou égal à 0.")

    except:
        print("vous n'avez pas saisi un nombre!")

    finally:
        nombresaisi = False
```

Résultats dans la fenêtre Python shell :

```
============ RESTART: D:\Travail\ArdPyLab\Docs\Python\exception2.py ============
Saisissez un nombre : az
vous n'avez pas saisi un nombre!
Saisissez un nombre : 0
Le nombre saisi est inférieur ou égal à 0.
Saisissez un nombre : 15
Ce n'est pas un nombre premier.
>>>
============ RESTART: D:\Travail\ArdPyLab\Docs\Python\exception2.py ============
Saisissez un nombre : 13
Le nombre 13 est premier.
>>>
```

Remarque :

L'instruction **break** du bloc **try** permet, en l'absence d'erreur, de sortir de la boucle **while** et de finir le programme.

. Synthèse structure des scripts Python

Les affectations de variables, les structures conditionnelles et itératives, les gestions d'erreurs sont la base des scripts Python.

Voici un script qui résume tout ce qui a été vu jusqu'à présent. Dans ce programme, l'utilisateur va créer un inventaire de matériel visualisable et modifiable.

Inventaire.py - D:/Travail/ArdPyLab/Docs/Python/Inventaire.py (3.7.7)

File Edit Format Run Options Window Help

```
print("/////////////// INVENTAIRE MATERIEL ///////////////")

Inventaire = {}
Fin =False

print("appuyer sur A pour ajouter un matériel à l'inventaire:")
print("appuyer sur S pour supprimer un matériel de l'inventaire:")
print("appuyer sur V pour afficher la liste de matériel:")
print("appuyer sur Q pour quitter:")
print("////////////////////////////////////////////////")
print(" ")

while Fin == False:

    choix=input()

    if choix.upper() == "A":
        Matos=input("saisissez le type de matériel à ajouter à l'inventaire:")
        Quant=input("saisissez la quantité de ce matériel:")
        Inventaire[Matos]=Quant

    if choix.upper() == "V":
        for cle, valeur in Inventaire.items():
            print("{} : {}".format(cle, valeur))

    if choix.upper() == "S":
        element=input("saisissez l'élément à supprimer dans l'inventaire:")
        try:
            del Inventaire[element]
        except:
            print("L'élément que vous voulez supprimer n'existe pas!")

    if choix.upper() == "Q":
        print("Fin du programme")
        Fin = True
```

Résultats dans la fenêtre Python Shell :

```
========== RESTART: D:/Travail/ArdPyLab/Docs/Python/Inventaire.py ==========
/////////////// INVENTAIRE MATERIEL ///////////////
appuyer sur A pour ajouter un matériel à l'inventaire:
appuyer sur S pour supprimer un matériel de l'inventaire:
appuyer sur V pour afficher la liste de matériel:
appuyer sur Q pour quitter:
////////////////////////////////////////////////

A
saisissez le type de matériel à ajouter à l'inventaire:Bécher
saisissez la quantité de ce matériel:20
a
saisissez le type de matériel à ajouter à l'inventaire:Eprouvette
saisissez la quantité de ce matériel:15
v
Bécher : 20
Eprouvette : 15
s
saisissez l'élément à supprimer dans l'inventaire:Bécher
v
Eprouvette : 15
a
saisissez le type de matériel à ajouter à l'inventaire:Erlenmeyer
saisissez la quantité de ce matériel:10
v
Eprouvette : 15
Erlenmeyer : 10
s
saisissez l'élément à supprimer dans l'inventaire:bécher
L'élément que vous voulez supprimer n'existe pas!
q
Fin du programme
```