```
#!coding: utf-8

######################
# Base TROIS : while #
######################

# les while sont comme des if, sauf qu'après avoir exécuté leur code, ils
recommencent

a = 5

if a < 10:
    print "Tada"

while a < 10:
    print "Hello"

# ce programme va afficher "Hello" sans s'arrêter...
# pour éviter de faire une "boucle infinie", il faudrait que la condition
devienne fausse
# par exemple comme ceci

i = 0
while i < 10:
    print "Hello"
    i = i + 1

# maintenant, Hello n'est affiché que dix fois ^^

# on peut bien sûr utiliser nos variables

i = 0
while i < 10:
    print "Hello", 2 * i + 5
    i = i + 1

# affichera "Hello 5", "Hello 7", "Hello 9" etc.

#########################
# Base QUATRE : listes  #
#########################

# on peut créer des listes, avec des crochets
ma_liste = [1,2,7,2] # 4 éléments !
# 4 opérations de base sont possibles sur les listes

# 1) Lire
premier = ma_liste[0] # l'élément numéro 0 est le premier
dernier = ma_liste[3] # vu que notre liste est de taille 4, 3 est le dernier
taille = len(ma_liste) # len permet de savoir la taille
print ma_liste[5] # ERREUR, il n'y a pas d'élément "5", le dernier était "3"

# le "numéro" est appelé "l'indice", l'indice du "1" dans notre liste est donc 0

# 2) Écrire
ma_liste[0] = 9 # hop ! la liste vaut [9,2,7,2]
ma_liste[5] = 2 # ERREUR : IndexError

# 3) Ajouter à la fin
ma_liste.append(0) # hop ! la liste vaut [9,2,7,2,0]
```

```
# 4) Supprimer
del ma_liste[1] # l'élément numéro "1" est supprimé, la liste vaut donc
[9,7,2,0]

# Les boucles et les listes vont bien ensemble,
# on peut par exemple afficher tous les éléments d'une liste avec un while

i = 0
while i < len(ma_liste):
    print ma_liste[i]
    i = i + 1

# Tu as toutes les bases pour faire tous les exercices
# Je conseille de faire au moins le 5 (max list) et puis de passer à pygame pour
créer des fenêtres !
# Deux approches sont possibles pour commencer pygame :
# - Soit tu lances pygame0_code_minimal.py et essaie de comprendre|modifier en
lisant les commentaires
# - Soit tu fais pas à pas les petits codes pygame1_dessin.py, pygame2_tick.py,
pygame3_events.py, pygame4_animations.py qui expliquent séparément les concepts
du code minimal
# Ensuite tu peux essayer de faire ton projet et lire les chapitres suivants
quand nécessaire (clavier, souris, images)

##########################
# Pour en savoir plus... #
##########################

### while ###
# certaines structures "while" sont très utilisées, et peuvent être remplacées
par des "for"

# compter ! avec range
for i in range(5):
    print i

# ce code peut s'écrire comme ceci avec un while :
i = 0
while i < 5:
    print i
    i = i + 1

# on peut aussi demander de ne pas commencer à 0

for i in range(1,5):
    print i

# ce qui peut s'écrire comme ceci :

i = 1
while i < 5:
    print i
    i = i + 1

# parcourir une liste ! sans savoir l'index

for nombre in ma_liste:
    print nombre

# ce qui peut s'écrire comme ceci avec un while

i = 0
while i < len(ma_liste):
    nombre = ma_liste[i]
    print nombre
```

```
    i = i + 1

# attention, il existe quelques subtiles différences,
# en cas de doute, utiliser while

### list ###

# Indices négatifs
ma_liste = [5,2,1,3]
print ma_liste[-1] # 3

# D'autres structures ressemblent aux list
# Mais la list est la plus générale : elle peut tout faire !

## le tuple
# on ne peut que le lire (donc uniquement l'opération 1)
mon_tuple = (1,2,3)
a = mon_tuple[0] # Lire
x = len(mon_tuple) # Lire
# Pas écrire, pas append, pas del :(

# un tuple peut même se créer sans les parenthèses
mon_tuple = 1,2,3 # stylé
# un tuple de taille 0 s'écrit ()
mon_tuple = ()
# un tuple de taille 1 doit avoir une virgule
mon_tuple = (1,)
mon_tuple = 1,

## unpacking
liste = [1,2,3]
a,b,c = liste
# permet de lire en un coup toutes les valeurs d'une liste
# utilisable uniquement quand on connaît la taille de la liste (essayez a,b =
[1,2,3] ou a,b = [1])
# marche avec les listes, les tuples
a,b,c = [1,2,3]
a,b,c = (1,2,3)
a,b,c = 1,2,3 # version la plus cool !

## les str (string / chaînes de caractères)
prenom = "Robert"
prenom = 'Robert' # même chose

# on ne peut que les lire
lettre = prenom[0] # 'R'
taille = len(prenom) # 6
print len(lettre) # 1

# on peut faire de drôles de math avec !
nom_de_famille = "Vanden Eynde"
long_nom = prenom + " " + nom_de_famille # "Robert Vanden Eynde"
beaucoup_de_nom = prenom * 5 # "RobertRobertRobertRobertRobert"
very_funny = 'ha' * 3 # "hahaha"

# comparer, via l'alphabet
if prenom < 'Frederic':
    print "Avant Frederic"
else:
    print "Après ou est Frederic"

# Attention, majuscules, minuscules
print('A' == 'a') # False
print('a' < 'B') # False
petit_robert = prenom.lower() # "robert"
```

```python
grand_robert = prenom.upper() # "ROBERT"

# str et liste de str
caracs = list("Robert")
print caracs # ['R', 'o', 'b', 'e', 'r', 't']

liste = ['A', 'B', 'C']
chaine = ''.join(liste) # "ABC"
chaine_v = ', '.join(liste) # "A, B, C"

# str et conversions
texte = str(52) # "52"
nombre = int("23") # 23
virgule = float("41.25") # 41.25
list_of_str = " Hello World How Is Life  ".split() # ['Hello', 'World', 'How',
'Is', 'Life']
some_names = "Hello World;Bonjour le monde;Donkey Konga".split(';') # ['Hello
World', 'Bonjour le monde', 'Donkey Konga']
no_space = " \t  Yeyo Yayo \n ".strip() # "Yeyo Yayo"

# slicing
print prenom[1:4] # de 1 à 4 non compris : "obe"
print prenom[:3] # du début à 3 non compris : "Rob"
print prenom[4:] # de 4 à la fin : "rt"

## format : créer facilement une chaîne depuis un modèle

phrase = "Bonjour {}, vous avez {} ans".format("Bob", 25)
# cela permet de séparer le modèle, des données (ici Bob et 25)

# on peut utiliser des chiffres pour faire référence à une certaine donnée (on
commence à 0)
phrase = "Bonjour {0}, vous avez {1} ans. Aurevoir {0}".format("Bob", 25)

# ou des noms
phrase = "Bonjour {nom}, vous avez {age} ans. Aurevoir {nom}".format(nom="Bob",
age=25)

# n'hésitez pas à passer à la ligne !
longue_phrase = "Bonjour {nom}, vous avez {age}. Votre score est de
{score}.".format(
    nom = "Bob",
    age = 25,
    score = 24,
)

## comprehension list (programmation fonctionnelle)

# souvent, on a un code comme ceci :

L = [] # On crée une liste vide
for i in range(50): # on fait un for
    L.append(i*i) # et la seule instruction du for est L.append(...)

# Avec les comprehension list on peut faire ça "en une ligne" :
L = [i*i for i in range(50)]

# de même, s'il y a un if (sans else !)

L = []
for i in range(50):
    if 10 <= i <= 15:
        L.append(i*i)

# en une ligne :
```

```python
L = [i*i for i in range(50) if 10 <= i <= 15]

# on peut même avoir plusieurs for et if

L = []
for i in range(50):
    for j in range(10):
        if i != 0:
            if j % 2 == 5:
                for k in range(90):
                    L.append(i+j+k)

# en une ligne, mais on passe à la ligne, c'est plus joli
L = [i+j+k for i in range(50)
           for j in range(10)
           if i != 0
           if j % 2 == 5
           for k in range(90)]
```