3ème Année Licence

Cours 06Bionformatique

Boucles

Dr. Mohammed Mehdi Bouchene

Boucles

Lorsqu'une opération doit être répétée plusieurs fois, par exemple sur tous les éléments d'une liste, nous évitons de devoir taper (ou copier et coller) du code répétitif en créant une boucle. Il existe deux manières de créer des boucles en Python, la boucle **for** et la boucle **while**.

La boucle for

La boucle **for** en Python itère sur chaque élément d'une séquence (telle qu'une liste ou un tuple) dans l'ordre dans lequel ils apparaissent dans la séquence. Cela signifie qu'une variable (code dans l'exemple ci-dessous) est définie à chaque élément de la séquence de valeurs, et chaque fois que cela se produit, le bloc de code indenté est exécuté à nouveau.

```
codeList = ['NA06984', 'NA06985', 'NA06986', 'NA06989', 'NA06991']
for code in codeList:
    print(code)
```

Une boucle **for** peut parcourir les caractères individuels d'une chaîne de caractères :

```
dnaSequence = 'ATGGTGTTGCC'
for base in dnaSequence:
    print(base)
```

Et aussi sur les clés d'un dictionnaire :

```
rnaMassDict = {"G":345.21, "C":305.18, "A":329.21, "U":302.16}
for x in rnaMassDict:
    print(x, rnaMassDict[x])
```

Toutes les variables définies avant la boucle sont accessibles à l'intérieur de la boucle. Ainsi, par exemple, pour calculer la somme des éléments dans une liste de valeurs, nous pourrions définir le total initial à zéro et ajouter chaque valeur au total de la boucle :

```
total = 0
values = [1, 2, 4, 8, 16]
for v in values:
    total = total + v
    # total += v
    print(total)
```

Naturellement, nous pouvons combiner une boucle **for** avec une instruction **if**, en notant qu'il nous faut deux niveaux d'indentation, un pour la boucle externe et un autre pour les blocs conditionnels :

```
geneExpression = {
    'Beta-Catenin': 2.5,
    'Beta-Actin': 1.7,
    'Pax6': 0,
    'HoxA2': -3.2
}
for gene in geneExpression:
    if geneExpression[gene] < 0:
        print(gene, "is downregulated")
    elif geneExpression[gene] > 0:
        print(gene, "is upregulated")
    else:
        print("No change in expression of ", gene)
```

Exercices 1

1. Créez une séquence où chaque élément est une base individuelle d'ADN. Faites la séquence 15 bases de long.
2. afficher la longueur de la séquence.
3. Créez une boucle **for** pour afficher chaque base de la séquence sur une nouvelle ligne.

La boucle while

En plus de la boucle **for** qui opère sur une collection d'éléments, il existe une boucle **while** qui se répète simplement pendant qu'une instruction est évaluée à **True** et s'arrête lorsqu'elle est **False**. Notez que si l'expression testée n'évalue jamais à False, vous avez une « boucle infinie », ce qui n'est pas bon.

Dans cet exemple, nous générons une série de nombres en doublant une valeur après chaque itération, jusqu'à atteindre une limite :

```
value = 0.25
while value < 8:
    value = value * 2
    print(value)
print("final value:", value)
print(total)
```

Ce qui se passe ici, c'est que la valeur est doublée à chaque itération et qu'une fois atteinte, le test de **while** échoue (8 n'est pas inférieur à 8) et cette dernière valeur est conservée. Notez que si le test avait la valeur <= 8, nous obtiendrions un doublement supplémentaire et la valeur atteindrait 16.

Exercices 2

1. Réutilisez la séquence longue de 15 bases créée lors de l'exercice précédent, où chaque élément est une base individuelle de l'ADN.

2. Créez une boucle **while** similaire à celle ci-dessus qui commence à la troisième base de la séquence et sort toutes les trois bases jusqu'au 12ème.

Sauter et casser des boucles

Python a deux façons d'affecter le flux de la boucle **for** ou **while** à l'intérieur du bloc. L'instruction **continue** signifie que le reste du code du bloc est ignoré pour cet élément particulier de la collection, c'est-à-dire que vous passez à l'itération suivante. Dans cet exemple, les nombres négatifs ne sont pas inclus dans la somme :

```
values = [10, -5, 3, -1, 7]
total = 0
for v in values:
    if v < 0:
        continue # Skip this iteration
    total += v
print(total)
```

L'autre façon d'affecter une boucle est d'utiliser l'instruction **break**. Contrairement à l'instruction **continue**, cela entraîne immédiatement la fin de toutes les boucles et l'exécution reprend à l'instruction suivante après la boucle.

```
geneticCode = {'TAT': 'Tyrosine',  'TAC': 'Tyrosine',
            'CAA': 'Glutamine', 'CAG': 'Glutamine',
            'TAG': 'STOP'}
sequence = ['CAG','TAC','CAA','TAG','TAC','CAG','CAA']
for codon in sequence:
    if geneticCode[codon] == 'STOP':
        break           # Quit looping at this point
    else:
        print(geneticCode[codon])
```

Boucle des pièges

Un compteur interne est utilisé pour garder trace du prochain élément à utiliser, et il est incrémenté à chaque itération. Lorsque ce compteur a atteint la longueur de la séquence, la boucle se termine. Cela signifie que si vous supprimez l'élément en cours de la séquence, l'élément suivant sera ignoré (car il obtient l'index de l'élément en cours qui a été déjà traité). De même, si vous insérez un élément dans une séquence avant l'élément en cours, l'élément en cours sera traité à nouveau lors de la prochaine boucle. Cela peut conduire à de mauvais bugs qui peuvent être évités en faisant une copie temporaire en utilisant une tranche de la séquence entière.

** En boucle, ne modifiez jamais la collection ! ** Créez toujours une copie de celle-ci en premier.

Plus de boucle

Utiliser range ()

Si vous souhaitez parcourir une séquence numérique, vous pouvez combiner la fonction **range ()** et une boucle **for**.

```
print(list(range(10)))
print(list(range(5, 10)))
print(list(range(0, 10, 3)))
print(list(range(7, 2, -2)))
```

Boucle à travers les gammes

```
for x in range(8):
    print(x*x)
```

```
squares = []
for x in range(8):
    s = x*x
    squares.append(s)
    print(squares)
```

Utiliser enumerate ()

Avec une séquence,la fonction **enumerate** () vous permet de parcourir la séquence en générant un tuple contenant chaque valeur ainsi qu'un index correspondant.

```
letters = ['A','C','G','T']
for index, letter in enumerate(letters):
    print(index, letter)
```

```
numbered_letters = list(enumerate(letters))
print(numbered_letters)
```

Filtrage en boucles

```
city_pops = {
    'London': 8200000,
    'Cambridge': 130000,
    'Edinburgh': 420000,
    'Glasgow': 1200000
}
big_cities = []
for city in city_pops:
    if city_pops[city] >= 1000000:
        big_cities.append(city)
print(big_cities)
```

```
total = 0
for city in city_pops:
    total += city_pops[city]
print("total population:", total)
```

```
pops = list(city_pops.values())
print("total population:", sum(pops))
```

Chaîne de formatage

Construire des chaînes plus complexes à partir d'une combinaison de variables de types différents peut s'avérer fastidieux et vous souhaitez parfois avoir plus de contrôle sur la manière dont les valeurs sont interpolées dans une chaîne. Python fournit un mécanisme puissant pour formater les chaînes à l'aide de la fonction **.format ()**, qui utilise des "champs de remplacement" entourés d'accolades {}, qui commence par un nom de champ facultatif suivi d'un signe deux points **:** et se termine par une spécification de format.

Il y a beaucoup de ces spécificateurs, mais en voici 3 utiles :

- d: entier décimal
- f: nombre en virgule flottante
- s: chaîne de caractères

Vous pouvez spécifier le nombre de points décimaux à utiliser dans un nombre à virgule flottante avec, par exemple. .2f pour utiliser 2 décimales ou + .2f pour utiliser 2 décimales avec toujours montrer son signe associé.

```
print('{:.2f}'.format(0.4567))
```

```
geneExpression = {
    'Beta-Catenin': 2.5,
    'Beta-Actin': 1.7,
    'Pax6': 0,
    'HoxA2': -3.2
}
for gene in geneExpression:
    print('{:s}\t{:+.2f}'.format(gene, geneExpression[gene])) # s is optional
    # could also be written using variable names
    #print('{gene:s}\t{exp:+.2f}'.format(gene=gene, exp=geneExpression[gene]))
```

Exercices 3

1. Calculons le contenu en GC d'une séquence d'ADN. Utilisez la séquence de 15 bases créée pour les exercices ci-dessus. Créez une variable, gc, que nous utiliserons pour compter le nombre de G ou C dans notre séquence. Afficher chaque base de la séquence à côté de son index sur une nouvelle ligne.

2. Créez une boucle pour parcourir les bases de votre séquence. Si la base est un G ou si la base est un C, ajoutez-en un à votre variable gc.

3. Lorsque la boucle est terminée, divisez le nombre de bases GC par la longueur de la séquence et multipliez-le par 100 pour obtenir le pourcentage GC. Formatez le résultat pour n'afficher que 2 décimales.