3ème Année Licence Technologie Agroalimentaire et Contrôle deQualité

TP 03 Bionformatique Listes et chaînes de caractères

Dr. Mohammed Mehdi Bouchene

En plus des types de données de base que nous avons vu le TP précèdent, vous souhaiterez très généralement de stocker et de manipuler des collections de valeurs. Python dispose de plusieurs structures de données que vous pouvez utiliser à cette fin. L'idée générale est que vous pouvez placer plusieurs éléments dans une seule collection, puis faire référence à cette collection dans son ensemble. Celui que vous utiliserez dépendra du problème que vous essayez de résoudre.

Tuples

- Peut contenir n'importe quel nombre d'éléments
- Peut contenir différents types d'éléments
- Ne peut pas être modifié une fois créé (ils sont immuables)
- Les articles ont un ordre défini

Un tuple est créé à l'aide de crochets ronds entourant les éléments qu'il contient, avec des virgules séparant les éléments individuels.

```
a = (123, 54, 92) # tuple de 4 nombre entier
b = () # tuple vide
c = ("Ala",) # tuple d`une seul chaine de caractère (note the trailing ",")
d = (2, 3, False, "Arg", None) # a tuple de different type de valeur
print(a)
print(b)
print(c)
print(d)
```

Vous pouvez bien sûr utiliser des variables dans les tuples et d'autres structures de données

```
x = 1.2
y = -0.3
z = 0.9
t = (x, y, z)
print(t)
```

Les tuples peuvent être empaquetés et décompressés avec une syntaxe pratique. Le nombre de variables utilisées pour décompresser le tuple doit correspondre au nombre d'éléments du tuple.

```
t = 2, 3, 4 # empaquetement d`un tuple
print('t est', t)
x, y, z = t # tuple unpacking
print('x is', x)
print('y is', y)
print('z is', z)
```

Des listes

- Peut contenir n'importe quel nombre d'éléments
- Peut contenir différents types d'éléments
- Peut être modifié une fois créé (ils sont mutables)
- Les articles ont un ordre particulier

Les listes sont créées avec des crochets autour de leurs éléments

```
a = [1, 3, 9]
b = ["ATG"]
c = []
print(a)
print(b)
print(c)
```

Les listes et les tuples peuvent contenir une autre liste et des tuples, ou tout autre type de collection:

```
matrix = [[1, 0], [0, 2]]
print(matrix)
```

Vous pouvez convertir entre des tuples et des listes avec les fonctions de **tuple**, **list**. Notez que ceux-ci créent une nouvelle collection avec les mêmes éléments et ne modifient pas l'original.

```
a = (1, 4, 9, 16)     # A tuple of numbers
b = ['G','C','A','T'] # A list of characters
print(a)
print(b)
l = list(a)   # Make a list based on a tuple
print(l)
t = tuple(b)  # Make a tuple based on a list
print(t)
```

Manipuler des tuples et des listes

Une fois que vos données sont dans une liste ou un tuple, Python vous permet d'accéder à des éléments de la liste et de les manipuler de différentes manières, par exemple en triant les données. Les tuples et les listes peuvent généralement être utilisés de manière très similaire.

Accès à l'index

Vous pouvez accéder à des éléments individuels de la collection à l'aide de leur index. Notez que le premier élément est à l'index 0. Les index négatifs comptent à rebours à partir de la fin.

```
t = (123, 54, 92, 87, 33)
x = [123, 54, 92, 87, 33]
print('t is', t)
print('t[0] is', t[0])
print('t[2] is', t[2])
print('x is', x)
print('x[-1] is', x[-1])
```

Les tranches

Vous pouvez également accéder à une gamme d'éléments, appelés tranches, à partir de listes internes et de tuples à l'aide de deux points **:** pour indiquer le début et la fin de la tranche entre crochets. Notez que la notation de tranche [a: b] inclut les positions allant de a à b, non compris.

```
t = (123, 54, 92, 87, 33)
x = [123, 54, 92, 87, 33]
print('t[1:3] is', t[1:3])
print('x[2:] is', x[2:])
print('x[:-1] is', x[:-1])
```

En opérateur

Vous pouvez vérifier si une valeur est dans un tuple ou une liste avec l'opérateur **in**, et vous pouvez le nier avec **not in**

```
t = (123, 54, 92, 87, 33)
x = [123, 54, 92, 87, 33]
print('123 in', x, 123 in x)
print('234 in', t, 234 in t)
print('999 not in', x, 999 not in x)
```

Fonctions len() et count()

Vous pouvez obtenir la longueur d'une liste ou d'un tuple avec la fonction intégrée **len()**, et vous pouvez compter le nombre d'éléments particuliers contenus dans une liste avec la fonction **count()**.

```
t = (123, 54, 92, 87, 33)
x = [123, 54, 92, 87, 33]
print("length of t is", len(t))
print("number of 33s in x is", x.count(33))
```

Modifier les listes

Vous pouvez modifier les listes en place, mais pas les tuples.

```
x = [123, 54, 92, 87, 33]
print(x)
x[2] = 33
print(x)
```

Les tuples ne peuvent pas être modifiés une fois qu'ils ont été créés. Si vous essayez de le faire, vous obtiendrez une erreur.

```
t = (123, 54, 92, 87, 33)
print(t)
t[1] = 4
```

Vous pouvez ajouter des éléments à la fin d'une liste avec la méthode **append()**

```
x = [123, 54, 92, 87, 33]
x.append(101)
print(x)
```

Ou insérer des valeurs à une certaine position avec la méthode **insert()**, en fournissant la position souhaitée ainsi que la nouvelle valeur

```
x = [123, 54, 92, 87, 33]
x.insert(3, 1111)
print(x)
```

et supprimer des valeurs par index avec **del**

```
x = [123, 54, 92, 87, 33]
print(x)
del x[0]
print(x)
```

Il est souvent utile de pouvoir combiner des tableaux, ce qui peut être fait avec la méthode **extend()** (comme la methode **append** ajouterait la liste complète en tant qu'élément unique dans la liste)

```
a = [1,2,3]
b = [4,5,6]
a.extend(b)
print(a)
a.append(b)
print(a)
```

Le symbole plus + est un raccourci pour l'opération d'extension lorsqu'il est appliqué aux listes :

```
a = [1, 2, 3]
b = [4, 5, 6]
a = a + b
print(a)
```

La syntaxe slice peut-être utilisée à gauche d'une opération d'affectation pour affecter des sous-régions d'une liste.

```
a = [1, 2, 3, 4, 5, 6]
a[1:3] = [9, 9, 9, 9]
print(a)
```

Vous pouvez changer l'ordre des éléments dans une liste

```
a = [1, 3, 5, 4, 2]
a.reverse()
print(a)
a.sort()
print(a)
```

Notez que tous les deux changent la liste, si vous voulez une copie triée de la liste tout en laissant l'original inchangé, utilisez la fonction **sorted()**

Exercice 1

Créez une liste de codons d'ADN pour la séquence protéique CLYSY en fonction des variables de codon que vous avez définies précédemment.

1. Afficher la séquence d'ADN de la protéine sur l'écran.
2. Afficher le codon ADN du dernier acide aminé de la séquence protéique.
3. Créez deux autres variables contenant la séquence d'ADN d'un codon d'arrêt et d'un codon de départ, remplacez le premier élément de la séquence d'ADN par le codon de départ et ajoutez le codon d'arrêt à la fin de la séquence d'ADN. Afficher la séquence d'ADN obtenue

Manipulations des chaines de caractères

Les chaines de caractères ressemblent beaucoup à des tuples de caractères, et des caractères individuels et des sous-chaînes peuvent être consultés et manipulés à l'aide d'opérations similaires à celles décrites précédemment.

```
text = "ATGTCATTTGT"
print(text[0])
print(text[-2])
print(text[0:6])
print("ATG" in text)
print("TGA" in text)
print(len(text))
```

Comme avec les tuples, essayer d'attribuer une valeur à un élément d'une chaîne entraîne une erreur

```
text = "ATGTCATTTGT"
text[0:2] = "CCC"
```

Python fournit un certain nombre de fonctions utiles qui vous permettent de manipuler des chaînes de caractères. L'opérateur **in** vous permet de vérifier si une sous-chaîne est contenue dans une chaîne de caractères plus grande, mais il ne vous dit pas où se trouve la sous-chaîne. Cela est souvent utile de le savoir et Python fournit la méthode **.find ()** qui renvoie l'index de la première occurrence de la chaîne de recherche et la méthode **.rfind ()** pour lancer la recherche à partir de la fin de la chaîne.

Si la chaîne de recherche ne figure pas dans la chaîne, ces deux méthodes renvoient -1.

```
dna = "ATGTCACCGTTT"
index = dna.find("TCA")
print("TCA is at position:", index)
index = dna.rfind('C')
print("The last Cytosine is at position:", index)
print("Position of a stop codon:", dna.find("TGA"))"
```

Lorsque nous lisons du texte à partir de fichiers (ce que nous verrons plus tard), il y a souvent des espaces non désirés au début ou à la fin de la chaîne. Nous pouvons supprimer les espaces en début de ligne avec la méthode **.lstrip ()**, les espaces en fin de chaîne avec **.rstrip ()** et les espaces des deux côtés avec **.strip ()**.

Toutes ces méthodes renvoient une copie de la chaîne modifiée. Par conséquent, si vous souhaitez remplacer l'original, vous pouvez affecter le résultat de l'appel de la méthode à la variable d'origine.

```
s = "    Chromosome Start End                "
print(len(s), s)
s = s.lstrip()
print(len(s), s)
s = s.rstrip()
print(len(s), s)
```

Vous pouvez diviser une chaîne en une liste de sous-chaînes à l'aide de la méthode **.split ()**, en fournissant le délimiteur en tant qu'argument à la méthode. Si vous ne fournissez pas de délimiteur, la méthode divisera la chaîne en blanc par défaut (ce qui est très souvent ce que vous voulez!)

Pour diviser une chaîne en caractères, vous pouvez simplement la convertir en liste par la fonction **list()**.

```
seq = "ATG TCA CCG GGC"
codons = seq.split(" ")
print(codons)


bases = list(seq) # a tuple of character converted into a list
print(bases)
```

.split () est la contrepartie de la méthode **.join ()** qui vous permet de joindre les éléments d'une liste à une chaîne uniquement si tous les éléments sont de type Chaîne:

```
seq = "ATG TCA CCG GGC"
codons = seq.split(" ")
print(codons)
print("|".join(codons))
```

Nous avons également vu précédemment que l'opérateur + vous permet de concaténer des chaînes en une chaîne plus grande.

```
s = "chr"
chrom_number = 2
print(s + str(chrom_number))print("|".join(codons))
```

Exercice 02

Créez une variable chaîne contenant votre nom complet, avec votre nom et prénom (ainsi que tous les prénoms) séparés par un espace. Divisez la chaîne en une liste et imprimez votre nom de famille.

Vérifiez si votre nom de famille contient la lettre "E" et imprimez la position de cette lettre dans la chaîne. Essayez quelques autres lettres.

1st base	2nd base								3rd base
	T		C		A		G		
T	TTT	(Phe/F) Phenylalanine	TCT	(Ser/S) Serine	TAT	(Tyr/Y) Tyrosine	TGT	(Cys/C) Cysteine	**T**
	TTC		TCC		TAC		TGC		**C**
	TTA	(Leu/L) Leucine	TCA		TAA	Stop (*Ochre*) [B]	TGA	Stop (*Opal*) [B]	**A**
	TTG		TCG		TAG	Stop (*Amber*) [B]	TGG	(Trp/W) Tryptophan	**G**
C	CTT		CCT	(Pro/P) Proline	CAT	(His/H) Histidine	CGT	(Arg/R) Arginine	**T**
	CTC		CCC		CAC		CGC		**C**
	CTA		CCA		CAA	(Gln/Q) Glutamine	CGA		**A**
	CTG		CCG		CAG		CGG		**G**
A	ATT	(Ile/I) Isoleucine	ACT	(Thr/T) Threonine	AAT	(Asn/N) Asparagine	AGT	(Ser/S) Serine	**T**
	ATC		ACC		AAC		AGC		**C**
	ATA		ACA		AAA	(Lys/K) Lysine	AGA	(Arg/R) Arginine	**A**
	ATG[A]	(Met/M) Methionine	ACG		AAG		AGG		**G**
G	GTT	(Val/V) Valine	GCT	(Ala/A) Alanine	GAT	(Asp/D) Aspartic acid	GGT	(Gly/G) Glycine	**T**
	GTC		GCC		GAC		GGC		**C**
	GTA		GCA		GAA	(Glu/E) Glutamic acid	GGA		**A**
	GTG		GCG		GAG		GGG		**G**