3ème Année Licence

Cours 05
Bionformatique
Exécution conditionnelle

Dr. Mohammed Mehdi Bouchene

Contrôle du programme et logique

Un programme sera normalement exécuté en exécutant les commandes indiquées, les unes après les autres, dans un ordre séquentiel. Cependant, vous aurez souvent besoin du programme pour vous en écarter. Il existe plusieurs façons de s'éloigner du paradigme ligne par ligne :

- Avec des déclarations conditionnelles. Ici, vous pouvez vérifier si une instruction ou une expression est vraie et si c'est le cas, continuez avec le bloc de code suivant, sinon vous pourriez le sauter ou exécuter un bit de code différent.
- En effectuant des boucles répétitives à travers le même bloc de code, où à chaque fois dans la boucle, différentes valeurs peuvent être utilisées pour les variables.
- Par l'utilisation de fonctions (sous-routines) où l'exécution du programme passe d'une ligne de code particulière à un emplacement totalement différent, même dans un fichier ou un module différent, pour effectuer une tâche avant (généralement) de revenir en arrière. Les fonctions sont abordées lors de la prochaine session, nous ne les discuterons pas encore.
- En vérifiant si une erreur ou une exception se produit, c'est-à-dire que quelque chose d'illégal est arrivé, et en exécutant différents blocs de code en conséquence Affichage des Valeurs

Blocs de code

Avec tous les moyens par lesquels l'exécution de code Python peut s'exercer, nous devons naturellement être conscients des limites du bloc de code dans lequel nous nous insérons, afin que le moment où le travail est terminé soit clairement défini et que l'exécution du programme puisse revenir en arrière. encore. En substance, il est nécessaire que la fin d'une fonction, d'une boucle ou d'une instruction conditionnelle soit définie afin de connaître les limites de leurs blocs de code respectifs.

Python utilise l'indentation pour indiquer les instructions contenues dans un bloc de code, tandis que d'autres langages utilisent des instructions de début et de fin spécifiques ou des accolades {}. Peu importe l'indentation que vous utilisez, mais le bloc entier doit être cohérent, c'est-à-dire que si la première instruction est indentée de quatre espaces, le reste du bloc doit être indenté du même montant. Le guide de style Python recommande l'utilisation d'une indentation de 4 espaces. Utilisez des espaces plutôt que des tabulations, car différents

éditeurs affichent des caractères de tabulation de différentes largeurs. L'utilisation de l'indentation pour délimiter des blocs de code est illustrée de manière abstraite dans le schéma suivant :

Déclaration 1 :

Commande A - dans le bloc d'instruction 1

Commande B - dans le bloc d'instruction 1

Affirmation 2:

Commande C - dans le bloc de l'instruction 2

Commande D - dans le bloc de l'instruction 2

Commande E - retour dans le bloc d'instruction 1

Commande F - en dehors de tous les blocs d'instructions

Exécution conditionnelle

La déclaration if

Une instruction if conditionnelle est utilisée pour spécifier qu'un bloc de code ne doit être exécuté que si un test associé est confirmé. Une expression conditionnelle est évaluée à True. Cela peut également impliquer des vérifications subsidiaires utilisant l'instruction elif pour utiliser un autre bloc si l'expression précédente s'avère être False. Il peut même y avoir une dernière déclaration à faire si aucune des vérifications n’est passée.

Ce qui suit utilise des instructions qui testent si un nombre est inférieur à zéro, supérieur à zéro ou autrement égal à zéro et afficheront un message différent dans chaque cas :

```
x = -3

if x > 0:
  print("Value is positive")

elif x < 0:
  print("Value is negative")
```

La forme générale d'écriture de ces déclarations conditionnelles combinées est la suivante :

```
if conditionnlExpression1:
    # codeBlock1
elif conditionnelExpression2:
    # codeBlock2
/
/
/
elif conditionnelExpressionN:
    # codeBlockN
    + n'importe quel nombre de déclarations elif supplémentaires, puis enfin:
Else :
    # codeBlockE
```

Le bloc **elif** est facultatif et nous pouvons en utiliser autant que nous le souhaitons. Le bloc else est également facultatif, il ne contiendra donc que l'instruction if, ce qui est une situation assez courante. Il est souvent recommandé d'inclure d'autres éléments lorsque cela est possible, afin de toujours détecter les cas qui ne

passent pas, sinon les valeurs risquent de passer inaperçues, ce qui pourrait ne pas être le comportement souhaité. Des espaces réservés sont nécessaires pour les blocs de code « vides»:

```
gene = "BRCA2"
geneExpression = -1.2

if geneExpression < 0:
    print(gene, "is downregulated")

elif geneExpression > 0:
    print(gene, "is upregulated")
```

Pour les vérifications conditionnelles très simples, vous pouvez écrire l'instruction **if** sur une seule ligne sous la forme d'une expression unique. Le résultat sera l'expression précédant if si la condition est vraie ou l'expression après l'autre.

```
x = 11
if x < 10:
    s = "Yes"
else:
    s = "No"
print(s)
# Could also be written onto one line
s = "Yes" if x < 10 else "No"
print(s)
```

Comparaisons et vérité

Avec l'exécution conditionnelle, la question se pose naturellement de savoir quelles expressions sont considérées comme vraies et quelles expressions sont fausses. Pour les valeurs booléennes python Vrai et Faux, la réponse est (espérons-le) évidente. En outre, les états logiques de vérité et de mensonge résultant de vérifications conditionnelles telles que «x est-il supérieur à 5?» Ou «y est-il dans cette liste?» Sont également clairs. Lors de la comparaison de valeurs, Python utilise les opérateurs de comparaison standard (ou relationnels), dont certains ont déjà été vus:

Opérateur	La description	Exemple
==	Égalité	1 == 2 # False
!=	Non égalité	1 != 2 # True
<	Moins que	1 < 2 # True
<=	Inférieur ou Égal à	2 <= 2 # True
>	Plus grand que	1 > 2 # False
>=	Supérieur ou égal à	1 >= 1 # True

Il est à noter que les opérations de comparaison peuvent être combinées, par exemple pour vérifier si une valeur est comprise dans une plage.

```
x = -5
if x > 0 and x < 10:
   print("In range A")


elif x < 0 or x > 10:
   print("In range B")else:
   pass
else:
  print("Value is zero")
```

Python a deux opérateurs de comparaison supplémentaires **`is`** et **`is not`**. Ceux-ci comparent si deux objets sont le même objet, alors que == et ! = comparent si les valeurs sont identiques. Il existe une règle simple pour dire quand utiliser == ou **`is`** :

- == est pour l'égalité des valeurs. Utilisez-le pour vérifier si deux objets ont la même valeur.
- est pour l'égalité de référence. Utilisez-le pour vérifier si deux références font référence au même objet.

Une chose à noter, vous obtiendrez des résultats inattendus et incohérents si vous utilisez par erreur est de comparer pour l'égalité de référence sur des entiers :

```
a = 500
b = 500
print(a == b) # True
print(a is b) # False
```

Un autre exemple avec les listes x, y et z:

- y étant une copie de x
- et z étant une autre référence au même objet que x

```
x = [123, 54, 92, 87, 33]
y = x[:] # y is a copy of x
z = x
print(x)
print(y)
print(z)
print("Are values of y and x the same?", y == x)
print("Are objects y and x the same?", y is x)
```

```
# Let's change x

x[1] = 23

print(x)

print(y)

print(z)

print("Are values of y and x the same?", y == x)

print("Are objects y and x the same?", y is x)

print("Are values of z and x the same?", z == x)
```

En Python, même les expressions qui n'impliquent pas une valeur booléenne évidente peuvent se voir attribuer le statut "vérité"; la valeur d'un élément lui-même peut être forcée à être considérée comme vraie ou fausse dans une instruction if. Pour les types intégrés Python décrits dans ce chapitre, les éléments suivants sont considérés comme faux dans un tel contexte :

False value	**La description**
None	numeric equality
False	False boolean
0	0 integer
0.0	0.0 floating point
""	empty string
()	empty tuple

Et tout le reste est considéré comme vrai dans un contexte conditionnel.

```
# Let's change x

x[1] = 23

print(x)

print(y)

print(z)

print("Are values of y and x the same?", y == x)

print("Are objects y and x the same?", y is x)

print("Are values of z and x the same?", z == x)
```

Exercices 1

Créez un bloc if..elif..else qui comparera une variable contenant votre âge à une autre variable contenant l'âge d'une autre personne et afficher une déclaration indiquant si vous êtes plus jeune, plus âgé ou du même âge que cette personne.

Exercices 2

Utilisez une instruction if pour vérifier si une variable contenant une séquence d'ADN contient un codon stop. (par exemple, dna = "ATGGCGGTCGAATAG"), commencez par vérifier un arrêt possible, puis étendez votre code pour rechercher l'un des trois codons d'arrêt (TAG, TAA, TGA).

Astuce: rappelez-vous que l'opérateur in vous permet de vérifier si une chaîne contient une sous-chaîne et renvoie True ou False en conséquence.