6 Tests

6.1 Définition

Les **tests** sont un élément essentiel à tout langage informatique si on veut lui donner un peu de complexité car ils permettent à l'ordinateur de prendre des décisions. Pour cela, Python utilise l'instruction `if` ainsi qu'une comparaison que nous avons abordée au chapitre précédent. Voici un premier exemple :

```
>>> x = 2
>>> if x == 2:
...     print("Le test est vrai !")
...
Le test est vrai !
```

et un second :

```
>>> x = "souris"
>>> if x == "tigre":
...     print("Le test est vrai !")
...
```

Il y a plusieurs remarques à faire concernant ces deux exemples :

- Dans le premier exemple, le test étant vrai, l'instruction `print("Le test est vrai !")` est exécutée. Dans le second exemple, le test est faux et rien n'est affiché.
- Les blocs d'instructions dans les tests doivent forcément être indentés comme pour les boucles `for` et `while`. L'indentation indique la portée des instructions à exécuter si le test est vrai.
- Comme avec les boucles `for` et `while`, la ligne qui contient l'instruction `if` se termine par le caractère deux-points « `:` ».

6.2 Tests à plusieurs cas

Parfois, il est pratique de tester si la condition est vraie ou si elle est fausse dans une même instruction `if`. Plutôt que d'utiliser deux instructions `if`, on peut se servir des instructions `if` et `else` :

```
>>> x = 2
>>> if x == 2:
...     print("Le test est vrai !")
```

```
... else:
...     print("Le test est faux !")
...
Le test est vrai !
>>> x = 3
>>> if x == 2:
...     print("Le test est vrai !")
... else:
...     print("Le test est faux !")
...
Le test est faux !
```

On peut utiliser une série de tests dans la même instruction `if`, notamment pour tester plusieurs valeurs d'une même variable.

Par exemple, on se propose de tirer au sort une base d'ADN puis d'afficher le nom de cette dernière. Dans le code suivant, nous utilisons l'instruction `random.choice(liste)` qui renvoie un élément choisi au hasard dans une liste. L'instruction `import random` sera vue plus tard dans le chapitre 8 *Modules*, admettez pour le moment qu'elle est nécessaire.

```
>>> import random
>>> base = random.choice(["a", "t", "c", "g"])
>>> if base == "a":
...     print("choix d'une adénine")
... elif base == "t":
...     print("choix d'une thymine")
... elif base == "c":
...     print("choix d'une cytosine")
... elif base == "g":
...     print("choix d'une guanine")
...
choix d'une cytosine
```

Dans cet exemple, Python teste la première condition, puis, si et seulement si elle est fausse, teste la deuxième et ainsi de suite... Le code correspondant à la première condition vérifiée est exécuté puis Python sort du bloc d'instructions du `if`.

6.3 Importance de l'indentation

De nouveau, faites bien attention à l'indentation ! Vous devez être très rigoureux sur ce point. Pour vous en convaincre, exécutez ces deux exemples de code :

Code 1

```
nombres = [4, 5, 6]
for nb in nombres:
    if nb == 5:
        print("Le test est vrai")
        print(f"car la variable nb vaut {nb}")
```

Résultat :

```
Le test est vrai
car la variable nb vaut 5
```

Code 2

```
nombres = [4, 5, 6]
for nb in nombres:
    if nb == 5:
        print("Le test est vrai")
    print(f"car la variable nb vaut {nb}")
```

Résultat :

```
car la variable nb vaut 4
Le test est vrai
car la variable nb vaut 5
car la variable nb vaut 6
```

Les deux codes pourtant très similaires produisent des résultats très différents. Si vous observez avec attention l'indentation des instructions sur la ligne 5, vous remarquerez que dans le code 1, l'instruction est indentée deux fois, ce qui signifie qu'elle appartient au bloc d'instructions du test `if`. Dans le code 2, l'instruction de la ligne 5 n'est indentée qu'une seule fois, ce qui fait qu'elle n'appartient plus au bloc d'instructions du test `if`, d'où l'affichage de `car la variable nb vaut xx` pour toutes les valeurs de `nb`.

6.4 Tests multiples

Les tests multiples permettent de tester plusieurs conditions en même temps en utilisant des opérateurs booléens. Les deux opérateurs les plus couramment utilisés sont le **OU** et le **ET**. Voici un petit rappel sur le fonctionnement de l'opérateur **OU** :

Condition 1	Opérateur	Condition 2	Résultat
Vrai	OU	Vrai	Vrai
Vrai	OU	Faux	Vrai
Faux	OU	Vrai	Vrai
Faux	OU	Faux	Faux

et de l'opérateur **ET** :

Condition 1	Opérateur	Condition 2	Résultat
Vrai	ET	Vrai	Vrai
Vrai	ET	Faux	Faux
Faux	ET	Vrai	Faux
Faux	ET	Faux	Faux

En Python, on utilise le mot réservé `and` pour l'opérateur **ET** et le mot réservé `or` pour l'opérateur **OU**. Respectez bien la casse des opérateurs `and` et `or` qui, en Python, s'écrivent en minuscule. En voici un exemple d'utilisation :

```
>>> x = 2
>>> y = 2
>>> if x == 2 and y == 2:
...     print("le test est vrai")
...
le test est vrai
```

Notez que le même résultat serait obtenu en utilisant deux instructions `if` imbriquées :

```
>>> x = 2
>>> y = 2
>>> if x == 2:
...     if y == 2:
...         print("le test est vrai")
...
le test est vrai
```

Vous pouvez aussi tester directement l'effet de ces opérateurs à l'aide de `True` et `False` (attention à respecter la casse).

```
>>> True or False
True
```

Enfin, on peut utiliser l'opérateur logique de négation `not` qui inverse le résultat d'une condition :

```
>>> not True
False
>>> not False
True
```

```
>>> not (True and True)
False
```

6.5 Instructions `break` et `continue`

Ces deux instructions permettent de modifier le comportement d'une boucle (`for` ou `while`) avec un test.

L'instruction `break` stoppe la boucle.

```
>>> for i in range(5):
...     if i > 2:
...         break
...     print(i)
...
0
1
2
```

L'instruction `continue` saute à l'itération suivante, sans exécuter la suite du bloc d'instructions de la boucle.

```
>>> for i in range(5):
...     if i == 2:
...         continue
...     print(i)
...
0
1
3
4
```

6.6 Tests de valeur sur des *floats*

Lorsque l'on souhaite tester la valeur d'une variable de type *float*, le premier réflexe serait d'utiliser l'opérateur d'égalité comme :

```
>>> 1/10 == 0.1
True
```

Toutefois, nous vous le déconseillons formellement. Pourquoi ? Python stocke les valeurs numériques des *floats* sous forme de nombres flottants (d'où leur nom !), et cela mène à certaines limitations. Observez l'exemple suivant :

```
>>> (3 - 2.7) == 0.3
False
```

```
>>> 3 - 2.7
0.2999999999999998
```

Nous voyons que le résultat de l'opération `3 - 2.7` n'est pas exactement `0.3` d'où le `False` en ligne 2.

En fait, ce problème ne vient pas de Python, mais plutôt de la manière dont un ordinateur traite les nombres flottants (comme un rapport de nombres binaires). Ainsi certaines valeurs de *float* ne peuvent être qu'approchées. Une manière de s'en rendre compte est d'utiliser l'écriture formatée en demandant l'affichage d'un grand nombre de décimales :

```
>>> 0.3
0.3
>>> f"{0.3:.5f}"
'0.30000'
>>> f"{0.3:.60f}"
'0.299999999999999988897769753748434595763683319091796875000000'
>>> f"{3 - 2.7:.60f}"
'0.299999999999999822364316059974953532218933105468750000000000'
```

On observe que lorsqu'on tape `0.3`, Python affiche une valeur arrondie. En réalité, le nombre réel `0.3` ne peut être qu'approché lorsqu'on le code en nombre flottant. Il est donc essentiel d'avoir cela en tête lorsque l'on effectue un test.

Conseils

Pour les raisons évoquées ci-dessus, il ne faut surtout pas tester si un *float* est égal à une certaine valeur. La bonne pratique est de vérifier si un *float* est compris dans un intervalle avec une certaine précision. Si on appelle cette précision *delta*, on peut procéder ainsi :

```
>>> delta = 0.0001
>>> var = 3.0 - 2.7
>>> 0.3 - delta < var < 0.3 + delta
True
>>> abs(var - 0.3) < delta
True
```

Ici on teste si `var` est compris dans l'intervalle $0.3 \pm delta$. Les deux méthodes mènent à un résultat strictement équivalent :

- La ligne 3 est intuitive car elle ressemble à un encadrement mathématique.
- La ligne 5 utilise la fonction valeur absolue `abs()` et est plus compacte.

6.7 Exercices

Conseil : pour ces exercices, créez des scripts puis exécutez-les dans un *shell*.

6.7.1 Jours de la semaine

Constituez une liste `semaine` contenant le nom des sept jours de la semaine.

En utilisant une boucle, écrivez chaque jour de la semaine ainsi que les messages suivants :

- `Au travail` s'il s'agit du lundi au jeudi ;
- `Chouette c'est vendredi` s'il s'agit du vendredi ;
- `Repos ce week-end` s'il s'agit du samedi ou du dimanche.

Ces messages ne sont que des suggestions, vous pouvez laisser libre cours à votre imagination.

6.7.2 Séquence complémentaire d'un brin d'ADN

La liste ci-dessous représente la séquence d'un brin d'ADN :

```
["A", "C", "G", "T", "T", "A", "G", "C", "T", "A", "A", "C", "G"]
```

Créez un script qui transforme cette séquence en sa séquence complémentaire.

Rappel : la séquence complémentaire s'obtient en remplaçant A par T, T par A, C par G et G par C.

6.7.3 Minimum d'une liste

La fonction `min()` de Python renvoie l'élément le plus petit d'une liste constituée de valeurs numériques ou de chaînes de caractères. Sans utiliser cette fonction, créez un script qui détermine le plus petit élément de la liste `[8, 4, 6, 1, 5]`.

6.7.4 Fréquence des acides aminés

La liste ci-dessous représente une séquence d'acides aminés :

```
["A", "R", "A", "W", "W", "A", "W", "A", "R", "W", "W", "R", "A", "G"]
```

Calculez la fréquence des acides aminés alanine (A), arginine (R), tryptophane (W) et glycine (G) dans cette séquence.

6.7.5 Notes et mention d'un étudiant

Voici les notes d'un étudiant : 14, 9, 13, 15 et 12. Créez un script qui affiche la note maximum (utilisez la fonction `max()`), la note minimum (utilisez la fonction `min()`) et qui calcule la moyenne.

Affichez la valeur de la moyenne avec deux décimales. Affichez aussi la mention obtenue sachant que la mention est « passable » si la moyenne est entre 10 inclus et 12 exclus, « assez bien » entre 12 inclus et 14 exclus et « bien » au-delà de 14.

6.7.6 Nombres pairs

Construisez une boucle qui parcourt les nombres de 0 à 20 et qui affiche les nombres pairs inférieurs ou égaux à 10 d'une part, et les nombres impairs strictement supérieurs à 10 d'autre part.

Pour cet exercice, vous pourrez utiliser l'opérateur modulo `%` qui renvoie le reste de la division entière entre deux nombres et dont voici quelques exemples d'utilisation :

```
>>> 4 % 3
1
>>> 5 % 3
2
>>> 4 % 2
0
>>> 5 % 2
1
>>> 6 % 2
0
>>> 7 % 2
1
```

Vous remarquerez qu'un nombre est pair lorsque le reste de sa division entière par 2 est nul.

6.7.7 Conjecture de Syracuse (exercice +++)

La conjecture de Syracuse est une conjecture mathématique qui reste improuvée à ce jour et qui est définie de la manière suivante.

Soit un entier positif *n*. Si *n* est pair, alors le diviser par 2. Si il est impair, alors le multiplier par 3 et lui ajouter 1. En répétant cette procédure, la suite de nombres atteint la valeur 1 puis se prolonge indéfiniment par une suite de trois valeurs triviales appelée cycle trivial.

Jusqu'à présent, la conjecture de Syracuse, selon laquelle depuis n'importe quel entier positif la suite de Syracuse atteint 1, n'a pas été mise en défaut.

Par exemple, les premiers éléments de la suite de Syracuse si on prend comme point de départ 10 sont : 10, 5, 16, 8, 4, 2, 1...

Créez un script qui, partant d'un entier positif *n* (par exemple 10 ou 20), crée une liste des nombres de la suite de Syracuse. Avec différents points de départ (c'est-à-dire avec différentes valeurs de *n*), la conjecture de Syracuse est-elle toujours vérifiée ? Quels sont les nombres qui constituent le cycle trivial ?

Remarque

1. Pour cet exercice, vous avez besoin de faire un nombre d'itérations inconnu pour que la suite de Syracuse atteigne le chiffre 1 puis entame son cycle trivial. Vous pourrez tester votre algorithme avec un nombre arbitraire d'itérations, typiquement 20 ou 100, suivant votre nombre *n* de départ.
2. Un nombre est pair lorsque le reste de sa division entière (opérateur modulo `%`) par 2 est nul.

6.7.8 Attribution de la structure secondaire des acides aminés d'une protéine (exercice +++)

Dans une protéine, les différents acides aminés sont liés entre eux par une liaison peptidique. Les angles phi et psi sont deux angles mesurés autour de cette liaison peptidique. Leurs valeurs sont utiles pour définir la conformation spatiale (appelée « structure secondaire ») adoptée par les acides aminés.

Par exemples, les angles phi et psi d'une conformation en « hélice alpha » parfaite ont une valeur de -57 degrés et -47 degrés respectivement. Bien sûr, il est très rare que l'on trouve ces valeurs parfaites dans une protéine, et il est habituel de tolérer une déviation de $\pm$ 30 degrés autour des valeurs idéales de ces angles.

Vous trouverez ci-dessous une liste de listes contenant les valeurs des angles phi et psi de 15 acides aminés de la protéine 1TFE :

```
[[48.6, 53.4],[-124.9, 156.7],[-66.2, -30.8], \
[-58.8, -43.1],[-73.9, -40.6],[-53.7, -37.5], \
[-80.6, -26.0],[-68.5, 135.0],[-64.9, -23.5], \
[-66.9, -45.5],[-69.6, -41.0],[-62.7, -37.5], \
[-68.2, -38.3],[-61.2, -49.1],[-59.7, -41.1]]
```

Pour le premier acide aminé, l'angle phi vaut *48.6* et l'angle psi *53.4*. Pour le deuxième, l'angle phi vaut *-124.9* et l'angle psi *156.7*, etc.

En utilisant cette liste, créez un script qui teste, pour chaque acide aminé, s'il est ou non en hélice et affiche les valeurs des angles phi et psi et le message adapté *est en hélice* ou *n'est pas en hélice*.

Par exemple, pour les 3 premiers acides aminés :

```
[48.6, 53.4] n'est pas en hélice
[-124.9, 156.7] n'est pas en hélice
[-66.2, -30.8] est en hélice
```

D'après vous, quelle est la structure secondaire majoritaire de ces 15 acides aminés ?

Remarque

Pour en savoir plus sur le monde merveilleux des protéines, n'hésitez pas à consulter la page Wikipedia sur la structure secondaire des protéines.

6.7.9 Détermination des nombres premiers inférieurs à 100 (exercice +++)

Voici un extrait de l'article sur les nombres premiers tiré de l'encyclopédie en ligne wikipédia.

Un nombre premier est un entier naturel qui admet exactement deux diviseurs distincts entiers et positifs (qui sont alors 1 et lui-même). Cette définition exclut 1, qui n'a qu'un seul diviseur entier positif. Par opposition, un nombre non nul produit de deux nombres entiers différents de 1 est dit composé. Par exemple $6 = 2 \times 3$ *est composé, tout comme* $21 = 3 \times 7$*, mais 11 est premier car 1 et 11 sont les seuls diviseurs de 11. Les nombres 0 et 1 ne sont ni premiers ni composés.*

Déterminez les nombres premiers inférieurs à 100. Combien y a-t-il de nombres premiers entre 0 et 100 ? Pour vous aider, nous vous proposons plusieurs méthodes.

Méthode 1 (peu optimale mais assez intuitive)

Pour chaque nombre de 2 à 100, calculez le reste de la division entière (avec l'opérateur modulo `%`) depuis 1 jusqu'à lui-même. Si c'est un nombre premier, il aura exactement deux nombres pour lesquels le reste de la division entière est égal à 0 (1 et lui-même). Si ce n'est pas un nombre premier, il aura plus de deux nombres pour lesquels le reste de la division entière est égal à 0.

Méthode 2 (plus optimale et plus rapide, mais un peu plus compliquée)

Parcourez tous les nombres de 2 à 100 et vérifiez si ceux-ci sont composés, c'est-à-dire qu'ils sont le produit de deux nombres premiers. Pratiquement, cela consiste à vérifier que le reste de la division entière (opérateur modulo `%`) entre le nombre considéré et chaque nombre premier déterminé jusqu'à maintenant est nul. Le cas échéant, ce nombre n'est pas premier. Attention, pour cette méthode, il faudra initialiser la liste de nombres premiers avec le premier nombre premier (donc 2 !).

6.7.10 Recherche d'un nombre par dichotomie (exercice +++)

La recherche par dichotomie est une méthode qui consiste à diviser (en général en parties égales) un problème pour en trouver la solution. À titre d'exemple, voici une discussion entre Pierre et Patrick dans laquelle Pierre essaie de deviner le nombre (compris entre 1 et 100 inclus) auquel Patrick a pensé.

- [Patrick] « C'est bon, j'ai pensé à un nombre entre 1 et 100. »

- [Pierre] « OK, je vais essayer de le deviner. Est-ce que ton nombre est plus petit ou plus grand que 50 ? »
- [Patrick] « Plus grand. »
- [Pierre] « Est-ce que ton nombre est plus petit, plus grand ou égal à 75 ? »
- [Patrick] « Plus grand. »
- [Pierre] « Est-ce que ton nombre est plus petit, plus grand ou égal à 87 ? »
- [Patrick] « Plus petit. »
- [Pierre] « Est-ce que ton nombre est plus petit, plus grand ou égal à 81 ? »
- [Patrick] « Plus petit. »
- [Pierre] « Est-ce que ton nombre est plus petit, plus grand ou égal à 78 ? »
- [Patrick] « Plus grand. »
- [Pierre] « Est-ce que ton nombre est plus petit, plus grand ou égal à 79 ? »
- [Patrick] « Égal. C'est le nombre auquel j'avais pensé. Bravo ! »

Pour arriver rapidement à deviner le nombre, l'astuce consiste à prendre à chaque fois la moitié de l'intervalle dans lequel se trouve le nombre. Voici le détail des différentes étapes :

1. le nombre se trouve entre 1 et 100, on propose 50 (100 / 2).
2. le nombre se trouve entre 50 et 100, on propose 75 (50 + (100-50)/2).
3. le nombre se trouve entre 75 et 100, on propose 87 (75 + (100-75)/2).
4. le nombre se trouve entre 75 et 87, on propose 81 (75 + (87-75)/2).
5. le nombre se trouve entre 75 et 81, on propose 78 (75 + (81-75)/2).
6. le nombre se trouve entre 78 et 81, on propose 79 (78 + (81-78)/2).

Créez un script qui reproduit ce jeu de devinettes. Vous pensez à un nombre entre 1 et 100 et l'ordinateur essaie de le deviner par dichotomie en vous posant des questions.

Votre programme utilisera la fonction `input()` pour interagir avec l'utilisateur. Voici un exemple de son fonctionnement :

```
>>> lettre = input("Entrez une lettre : ")
Entrez une lettre : P
>>> print(lettre)
P
```

Pour vous guider, voici ce que donnerait le programme avec la conversation précédente :

```
Pensez à un nombre entre 1 et 100.
Est-ce votre nombre est plus grand, plus petit ou égal à 50 ? [+/-/=] +
Est-ce votre nombre est plus grand, plus petit ou égal à 75 ? [+/-/=] +
Est-ce votre nombre est plus grand, plus petit ou égal à 87 ? [+/-/=] -
```

```
Est-ce votre nombre est plus grand, plus petit ou égal à 81 ? [+/-/=] -
Est-ce votre nombre est plus grand, plus petit ou égal à 78 ? [+/-/=] +
Est-ce votre nombre est plus grand, plus petit ou égal à 79 ? [+/-/=] =
J'ai trouvé en 6 questions !
```

Les caractères `[+/-/=]` indiquent à l'utilisateur comment il doit interagir avec l'ordinateur, c'est-à-dire entrer soit le caractère `+` si le nombre choisi est plus grand que le nombre proposé par l'ordinateur, soit le caractère `-` si le nombre choisi est plus petit que le nombre proposé par l'ordinateur, soit le caractère `=` si le nombre choisi est celui proposé par l'ordinateur (en appuyant ensuite sur la touche *Entrée*).