Programmer en Python
SVE Licence 1, Mathématiques
TP1 - Premiers pas

Pourquoi Python ?

Python est un langage interprété, c'est-à-dire que chaque ligne de code est lue puis interprétée afin d'être exécutée par l'ordinateur. Il est **gratuit** et on peut l'utiliser sans restriction dans des projets commerciaux. Python convient aussi bien à des **scripts** d'une dizaine de lignes qu'à des **projets complexes** de plusieurs dizaines de milliers de lignes.

Python présente la particularité de pouvoir être utilisé de plusieurs manières différentes. Vous allez d'abord l'utiliser en mode interactif, c'est-à-dire d'une manière telle que vous pourrez dialoguer avec lui directement depuis le clavier. Cela vous permettra de découvrir très vite un grand nombre de fonctionnalités du langage. Dans un second temps, vous apprendrez comment créer vos premiers programmes (scripts) et les sauvegarder sur disque.

Installation de Python

Nous allons maintenant vous expliquer comment installer cet environnement de programmation Thonny sur vos ordinateurs personnels (à la maison) :

1. Téléchargez `Thonny` pour Windows ici : `http://thonny.org/`.
2. Vous avez normalement téléchargé un fichier nommé `thonny-3.2.7.exe`. Ce fichier est un executable, executez-le en double cliquant dessus.
3. Lancez une première fois `Thonny` (un raccourci a certainement été crée sur votre bureau). Nous allons profiter de ce premier lancement pour télécharger deux modules très utiles, en utilisant le menu `Tools > Manage packages` :
   — installez le module `numpy`, qui permet de faire du calcul matriciel (entre autres) ;
   — installez le module `matplotlib`, qui permet de générer des graphiques et de tracer des courbes.

Si vous le souhaitez, vous pouvez installer `Thonny` sur vos ordinateurs personnels en suivant la même procédure. Bon travail !

1 Prise en main en mode interactif

Ouvrir une session Thonny (en utilisant le menu démarrer ou à l'aide de l'icone). Une interface s'ouvre qui présente un fenêtre de commande (en bas) et un éditeur de texte

(fenêtre du haut). . Le triple chevron >>> est l'invite de Python (prompt en anglais), ce qui signifie que Python attend une commande.

Par exemple, vous pouvez tout de suite utiliser l'interpréteur comme une simple calculatrice de bureau.

```
>>> 5+3
>>> 2 + 9 # les espaces sont optionnels
>>> 7 + 3 * 4 # la hiérarchie des opérations mathématiques
# est-elle respectée ?
>>> (7+3)*4
>>> 20 / 3
```

Vous pouvez aussi manipuler facilement des chaînes de caractères (ie. le texte)

```
>>>print("Bonjour!")
```

Remarque : L'interpréteur reconnait la chaîne de caractères grâce aux guillements qui l'entourent.

2 Variables

En informatique, on peut stocker des valeurs dans des variables définies par leur nom. Dans l'exemple ci-dessous, `x` et `codon_stop` sont deux variables (`x` stocke un nombre, et `codon_stop` stocke une chaîne de caractères) :

```
>>> x = 2
>>> print(x)
>>>
>>> codon_stop = "UAA"
>>> print(codon_stop)
>>>
>>> print(3*codon_stop)
>>>
>>> print("AGA"+codon_stop)
```

Attention à ne pas confondre le nom de la variable avec son contenu. Si on ajoute des paranthèse autour du nom ça devient une chaîne de caractère (voir exercice 1 ci-dessous).

L'intérêt de stocker les valeurs dans des variables est qu'on peut les réutiliser en faisant référence à la variable (et non plus à la valeur).

Le nom des variables en Python peut-être constitué de lettres minuscules (`a` à `z`), de lettres majuscules (`A` à `Z`), de chiffres (`0` à `9`) ou du caractère souligné (`_`). Néanmoins, un nom de variable ne doit pas débuter ni par un chiffre, ni par `_` et ne peut pas contenir de caractère accentué.

Exercice :

1. Que renvoie la liste d'instructions ci-dessous ?

```
>>> x = 2
>>> print("x")
>>> print(x)
>>> print(2*x+1)
>>> print(2*"x"+1)
```

2. Créer une variable de type entier `y` prenant la valeur 5.
   Afficher la valeur de `y`.
3. Créer une variable `nom` de type caractère contenant votre nom.
   Affichier la valeur de `nom`.
4. Créer une variable `nom` de type caractère contenant votre nom.
   Afficher le texte "Bonjour" suivi de votre nom (par exemple "Bonjour Paul") en utilisant la variable `nom`..
   Pensez à mettre une virgule ou un signe `+` entre "Bonjour" et `nom`.

3 Opérations

Opérations sur les chiffres

Les quatre opérations de base se font de manière simple sur les types numériques (nombres entiers et réels) :

```
>>> x = 45
>>> x + 2
47
>>> y = 2.5
>>> x + y
47.5
>>> (x * 10) / y
180.0
>>> x**2   # ** représente l'exposant
2025
```

Exercice : Le pourcentage de GC dans une séquence est calculé à l'aide de la formule

$$\frac{G+C}{A+T+G+C} * 100$$

où G désigne le nombre de guanines dans la séquence, A le nombre d'adénines, C celui de cytosines et T celui de thymines.

5. Créer les variables `A`, `C`,`G` et `T` et leur affecter respectivement les nombres d'adénines, de cytonies, de guanines et de thymines suivants : 4500 guanines, 2575 cytosines, 3025 adénines et 4700 thymines.

6. Calculez le pourcentage de GC avec le code pour une sequence contenant 4500 guanines, 2575 cytosines, 3025 adénines et 4700 thymines.
7. Calculez le pourcentage de AT pour la même séquence.

Opérations sur les chaînes de caractères
Deviner ce que font les instructions suivantes, puis les exécuter pour vérifier.

```
>>> chaine = "Salut"
>>> print(chaine)
>>> print(chaine + " Python")
>>> print(chaine * 3)
```

Exercice :

8. On appelle polyC un oligonucléotide qui ne contient que des bases C. Par exemple CCCCCC. Cet oligonucléotide est de longueur 6 car il contient 6 bases.
   Générez une chaîne de caractères représentant un oligonucléotide polyA (AAAA...) de 20 bases de longueur, sans taper littéralement toutes les bases (ie en utilisant les opérateurs `+` ou `*` pour les caractères).
   Pour stocker ce résultat en mémoire, on le "met" dans une variable dont on choisit le nom. Ici on choisira `polyA`. En langage informatique, on dit qu'on affecte la chaine à une variable.
   Vérifiez la longueur de la chaine obtenue à l'aide de la fonction `len`.

   ```
   len(polyA)
   ```

9. Suivant le modèle du dessus, générez en une ligne de code un polyA de 20 bases suivi d'un polyGC régulier (GCGCGC...) de 40 bases.

4 Tests et instructions conditionnelles

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 si telle ou telle condition est vraie ou fausse. Pour cela, Python utilise l'instruction `if` ainsi qu'une condition. Voici un exemple :

```
fumeur = input("patient fumeur ('oui' ou 'non') ? ")

if fumeur == "oui":
    niveau_de_risque = 3
else:
    niveau_de_risque = 0
print("Niveau de risque =", niveau_de_risque)
```

Explications du code ci-dessus : la fonction `input` *permet d'interagir avec l'utilisateur du code. Elle affiche à l'écran la chaîne de caractères entre guillemets (ici "patient fumeur ('oui' ou 'non') ? "). L'utilisateur doit alors taper sa réponse et la réponse est stockée dans la variable* `fumeur`.
Ensuite l'instruction `if` *teste la condition fumeur/non fumeur et affecte le niveau de risque à la variable* `risque` *en fonction du résultat du test. Finelement le niveau de risque est affiché dans la console par la fonction* `print`.

Remarquez l'**indentation** (ie. les décalages à droite au sein du texte). Elle est **indispensable** pour que Python identifie correctement les blocs d'instructions à considérer selon les conditions et interprète correctement les commandes.
Remarquez aussi le `==` qui permet de tester une égalité contrairement au `=` qui permet de faire des affectations.

Voici d'autres exemples de tests :

```
if niveau_de_risque == 0:
    print("Le risque est nul !")
if niveau_de_risque != 0:
    print("Il y a un risque !")
if niveau_de_risque >= 3:
    print("Risque élevé !")
```

De façon plus générale, l'instruction `if` s'utilise de la façon suivante

```
if condition1:
    code exécuté si condition1 est vraie ...
elif condition2:
    code exécuté si condition1 est fausse et condition2 est vraie...
else:
    code exécuté si condition1 et condition2 sont fausses
```

Les opérateurs de comparaison sont les suivants : `==` égal, `!=` différent, `>` supérieur, `>=` supérieur ou égal, `<` inférieur, `>=` inférieur ou égal.

Vous pouvez bien sûr cumuler plusieurs conditions en utilisant les opérateurs logiques ET, OU, NON. Par exemple :

```
if (fumeur == "oui") and (age > 60):
    print("le patient est une personne agee qui fume !")
if (fumeur == "oui") or (age > 60):
    print("le patient est une personne agee ou un fumeur!")
if not(fumeur == "oui"):
    print("le patient est non fumeur!")
```

Exercice :

10. Un service de photocopies facture 0,10 euros les 30 premières photocopies et 0,05 euros les suivantes.
Écrire les instructions qui donnent le montant payé après que le client a entré son nombre de photocopies. On pourra utiliser

```
nb = input("Entrez le nombre de photocopies ")
nb = int(nb)  # la sortie de input est une chaine de caractères,
              # la fonction int permet de la convertir
              # en nombre entier.

# écrire la suite ici pour calculer le prix en fonction de nb !
```

11. Écrire une suite d'instructions qui détermine si les racines d'un polynôme du second degré

$$ax^2 + bx + c$$

sont réelles ou non et qui renvoie la valeur des racines si elles sont réelles. On suivra les étapes suivantes

(a) Affecter des valeurs à a, b et c. Exemple :

```
a = -1
b = 4
c = 1
```

(b) Calculez le discriminant (et le stocker dans une variable).

(c) Écrire les instructions qui vérifient si le déterminant est positif et qui renvoie les valeurs des racines si c'est le cas et qui affiche "les racines ne sont pas réelles" sinon. Attention à la façon dont vous utilisez les parenthèses !

(d) On pourra prévoir un cas particulier si le discriminant est nul (ou très petit).

(e) Vérifier en faisant un calcul à la main que vous obtenez bien les bonnes racines.