3ème Année Licence Technologie Agroalimentaire et Contrôle de Qualité

# Cours 02 Bionformatique
# Notion des variables

Dr. Mohammed Mehdi Bouchene

## I. Organisation du cours

### Python

Le langage de programmation Python et son interpréteur Python (ligne de commande) Python est gratuit, multiplateforme, largement utilisé, bien documenté et bien pris en charge. Python est un langage interprété simple, sans étape de compilation séparée.

### Affichage des Valeurs

Utilisation de variables : ce sont des noms de valeurs, créés par l'utilisation. Aucune déclaration nécessaire. Une variable est juste un nom, elle n'a pas de type. Les valeurs sont collectées, si rien ne se réfère plus aux données, elles peuvent être recyclées. Doit assigner une valeur à la variable avant de l'utiliser. Python n'assume pas les valeurs par défaut pour les variables, ce qui peut masquer de nombreuses erreurs.

### Types de données simples

Les valeurs ont des types. Utilisez les fonctions pour convertir les types. Booléens, entiers, nombres à virgule flottante, nombres complexes, les chaînes de caractères. sont des séquences de caractères. Arithmétique: addition,soustraction, multiplication, division, exponentiation, reste. Enregistrer le code dans les fichiers, commentaires.

## II. Afficher des valeurs

Le premier élément de la syntaxe Python que nous allons apprendre est l'instruction **print**. Cette commande nous permet d'afficher des messages à l'utilisateur et de voir ce que Python considère comme la valeur d'une expression (très utile lors du débogage de vos programmes).

Nous entrerons dans les détails plus tard, mais pour le moment, notez que pour afficher du texte, vous devez le placer entre "guillemets".

Nous détaillerons sous peu les opérations arithmétiques prises en charge par python, mais vous pouvez essayer d'explorer les capacités de calcul de python.

```
print("Bonjour python!")
print(34)
print(2 + 3)
```

Vous pouvez afficher plusieurs expressions dont vous avez besoin pour les séparer par des virgules. Python insérera un espace entre chaque élément et une nouvelle ligne à la fin du message (vous pouvez toutefois supprimer ce problème en laissant une virgule à la fin de la commande).

```
print("la réponse est:", 42)
```

**En utilisant des variables**

Dans les commandes d'affichage ci-dessus, nous avons directement utilisé des valeurs telles que chaînes de caractères et les nombres. Lors de la programmation, nous voudrons généralement traiter avec des expressions plutôt complexes, dans lesquelles il est utile de pouvoir attribuer un nom à une expression, en particulier si nous essayons de gérer plusieurs valeurs en même temps.

Nous pouvons donner un nom à une valeur en utilisant des variables, le nom est apt car les valeurs stockées dans une variable peuvent varier. Contrairement à d'autres langages, le type de valeur affecté à une variable peut également changer (c'est l'une des raisons pour lesquelles Python est appelé langage dynamique).

Une variable peut être assignée à une valeur simple

```
x = 3
print(x)
```

... ou le résultat d'une expression plus complexe.

```
x = 2 + 2
print(x)
```

Une variable peut être appelée comme vous voulez (tant qu'elle commence par un caractère, elle ne contient ni espace ni signification) et vous affectez une valeur à une variable avec l'opérateur =. Notez que ceci est différent de l'égalité mathématique (sur laquelle nous reviendrons plus tard ...)

Vous pouvez afficher une variable pour voir ce que Python pense être sa valeur actuelle.

```
serine = "TCA"
print(serine, "codes pour le serine")
serine = "TCG"
print("de même que ", serine)
```

Les variables peuvent également être utilisées du côté droit d'une affectation. Dans ce cas, elles seront évaluées avant que la valeur ne soit affectée à la variable du côté gauche.

```
x = 5
y = x * 3
print(y)
```

vous pouvez utiliser la valeur actuelle d'une variable elle-même dans une affectation

```
y = y + 1
```

**Exercice 01**

Créez une variable et attribuez-lui la valeur une chaîne de caractères de votre prénom, attribuez votre âge à une autre variable (vous êtes libre de mentir !), afficher un message indiquant votre nom et âge.

Utilisez l'opérateur d'addition pour ajouter 10 ans à votre âge et afficher un message indiquant votre âge dans 10 ans.

## III. Types de données simples

Python (et les ordinateurs en général) traite différents types de données différemment. Python a 4 principaux types de données de base. Les types sont utiles pour contraindre certaines opérations à une certaine catégorie de variables. Par exemple, essayer de diviser une chaîne n'a pas vraiment de sens.

**Entiers**

```
i = -7
j = 123
print(i, j)
```

**nombre en virgule flottante**

un nombre représenté en virgule flottante, sont des nombres exprimés dans le système décimal, c'est-à-dire 2.1, 999.998, -0.000004.

```
x = 3.14159
y = -42.3
print(x * y)
```

Les nombres en virgule flottante peuvent également porter un suffixe e qui indique à quelle puissance ils opèrent.

```
k = 1.5e3
l = 3e-2
print(k)
print(l)
```

**Chaînes de caractères**

Les chaînes de caractères représentent du texte, c'est-à-dire des "chaînes" de caractères. Ils peuvent être délimités par des guillemets simples ou doubles, mais vous devez utiliser le même délimiteur aux deux extrémités. Contrairement à certains langages de programmation, tels que Perl, il n'y a pas de différence entre les deux types de guillemets, bien que l'utilisation d'un type permette à l'autre type d'apparaître dans la chaîne en tant que caractère normal.

Normalement, une instruction Python se termine à la fin de la ligne, mais si vous souhaitez taper une chaîne sur plusieurs lignes, vous pouvez la placer entre guillemets triples

```
s = "ATGTCGTCTACAACACT"
t = 'Serine'
u = "c`est une chaîne de caractères "
v = """A string that extends
over multiple lines"""
print(v)
```

**Booléens**

Les valeurs booléennes représentent la vérité ou le mensonge, tel qu'il est utilisé dans les opérations logiques, par exemple. Sans surprise, il n'y a que deux valeurs, et en Python, elles sont appelées True et False.

```
a = True
b = False
print(a, b)
```

**Type d'objet**

Vous pouvez vérifier quel type Python pense qu'une expression est avec la fonction **type**, que vous pouvez appeler avec le nom type immédiatement suivi de parenthèses entourant l'expression que vous souhaitez vérifier (une variable ou une valeur), par exemple. type (3). (Ceci est la forme générale pour appeler des fonctions, nous verrons beaucoup plus d'exemples de fonctions plus tard ...)

```
a = True
print(a, "is of", type(a))
```

```
i = -7
print(i, "is of", type(i))
```

```
x = 12.7893
print(x, "is of", type(x))
```

```
s = "ATGTCGTCTACAACACT"
print(s, "is of", type(s))
```

**Commentaires**

Lorsque vous écrivez un programme, il est souvent pratique d'annoter votre code pour vous rappeler ce que vous aviez (l'intention) de le faire. En programmation, ces annotations sont appelées commentaires. Vous pouvez inclure un commentaire en Python en préfixant du texte avec un caractère #. Tout le texte suivant le # sera alors ignoré par l'interpréteur python. Vous pouvez commencer un commentaire sur sa propre ligne ou l'inclure à la fin d'une ligne de code.

```
print("Hi") # this will be ignored
# as will this
print("Bye")
# print "Never seen"
```

**Arithmétique**

Python supporte toutes les opérations arithmétiques standard sur les types numériques et utilise la plupart du temps une syntaxe similaire à celle de plusieurs autres langages informatiques:

```
x = 4.5
y = 2
print('x', x, 'y', y)
print('addition x + y =', x + y)
print('Soustraction x - y =', x - y)
print('multiplication x * y =', x * y)
print('division x / y =', x / y)
```

```
x = 4.5
y = 2
print('x', x, 'y', y)
print('division x / y =', x / y)
print('floored division x // y =', x // y)
print('modulus (remainder of x/y) x % y =', x % y)
print('exponentiation x ** y =', x ** y)
```

Vous pouvez forcer Python à utiliser un type particulier en convertissant une expression de manière explicite, à l'aide de fonctions nommées utiles: float, int, str, etc.

```
float(3) + float(7)
int(3.14159) + 1
```

Comme d'habitude en maths, la division et la multiplication ont une priorité supérieure à celle de l'addition et de la soustraction, mais les expressions arithmétiques peuvent être regroupées à l'aide de parenthèses pour remplacer la priorité par défaut.

L'opérateur d'addition + vous permet également de concaténer des chaînes.

```
print('number' + str(3))
```

**Exercises 02**

Attribuez des valeurs numériques à 2 variables, calculez la moyenne de ces deux variables et stockez le résultat dans une autre variable. Imprimez le résultat à l'écran.

**Exercices 03**

Créez un nouveau fichier Python pour résoudre ces exercices. Il est recommandé de créer un nouveau fichier chaque fois que vous résolvez un nouveau problème.

Recherchez le code génétique. Créez quatre variables de chaîne qui stockent les codages ADN possibles de la sérine (S), de la leucine (L), de la tyrosine (Y) et de la cystéine (C). Lorsque plusieurs codages sont disponibles, choisissez-en un pour l'instant. Créez une variable contenant une séquence d'ADN possible pour la séquence protéique SYLYC. (Notez que l'opérateur d'addition + vous permet de concaténer des chaînes.) afficher la séquence d'ADN. Incluez un commentaire dans votre fichier pour vous rappeler l'objectif du script.

| 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 |