Logiciel de calcul formel

Loïc Demange
loic.demange@etud.univ-paris8.fr

5 février 2021

UNIVERSITÉ PARIS 8 VINCENNES-SAINT-DENIS

TP précédent

Objectifs du TP : avec l'aide de la documentation `https://doc.sagemath.org/html/fr/tutorial/tour_assignment.html`

- Attribuer une valeur à une variable, et multiplier cette variable par 2. Afficher le résultat.
- Attribuer une valeur à une variable, et multiplier cette variable par 2.01. Afficher et constater le type du résultat.
- Attribuer une valeur à une variable, et diviser cette variable par 2. Afficher uniquement sa partie entière si le nombre n'est pas pair.
- Attribuer une valeur à une variable, et réaliser l'opération modulo 2. Afficher le résultat.
- Afficher les 20 premières décimales de π.
- Afficher la racine carrée de 3. Afficher son approximation numérique à 10 décimales.
- Afficher le cosinus de 25. Afficher son approximation numérique à 10 décimales.
- Afficher le résultat de la comparaison entre deux variables.

TP précédent

Attribuer une valeur à une variable, et multiplier cette variable par 2. Afficher le résultat

```
a = 2
print(a*2)
```

TP précédent

Attribuer une valeur à une variable, et multiplier cette variable par 2.01. Afficher et constater le type du résultat.

```
a = 2
b = a * 2.01
print(b)
print(type(b))
```

TP précédent

Attribuer une valeur à une variable, et diviser cette variable par 2. Afficher uniquement sa partie entière si le nombre n'est pas pair.

```
a = 15
print(a//2)
```

TP précédent

Attribuer une valeur à une variable, et réaliser l'opération modulo 2. Afficher le résultat.

```
a = 15
print(a%2)
```

TP précédent

Afficher les 20 premières décimales de π.

```
print(n(pi, digits=20))
```

ou

```
print(numerical_approx(pi, digits=20))
```

TP précédent

Afficher la racine carrée de 3. Afficher son approximation numérique à 10 décimales.

```
print(n(sqrt(3), digits=10))
```

TP précédent

Afficher le cosinus de 25. Afficher son approximation numérique à 10 décimales.

```
print(n(cos(25), digits=10))
```

TP précédent

Afficher le résultat de la comparaison entre deux variables.

```
a = 10
b = 11
print(a == b)
```

Liste

En Python, une liste est une structure de données de taille flexible qui peut contenir n'importe quel type de données.
Comme un tableau, on peut accéder aux cases à l'aide des opérateurs **[]**.

Pour déclarer une liste, on peut le faire de cette façon.

```
l = [1,"salut", 's', 2.02]
```

On peut ajouter des éléments de plusieurs façons, les deux méthodes sont similaires.

```
l += [2]
l.append(2)
```

Conditionnel

Comme dans tout langage, on peut utiliser le conditionnel en adjoignant le mot-clé **if** avec une expression conditionnelle, ainsi que **:** pour commencer le bloc.
Comme Sage est tiré du langage Python, le sein de la condition sera délimité par et grâce à l'indentation.

Par exemple, voici une fonction qui ajoute 2 si la variable vaut 4.

```
a = 4

if a == 4:
    a = a + 2

print(a)
```

Remarque Vu que le **print** n'est pas indenté, il ne fait pas partie du conditionnel.

Conditionnel

On peut aussi ajouter un nombre infini de **Sinon Si** au bloc conditionnel grâce au mot-clé **elif**, ainsi qu'un **Sinon** avec **else**. Comme Sage est tiré du langage Python, le sein de la condition sera délimité par et grâce à l'indentation.

Par exemple, voici une fonction qui ajoute 2 si la variable vaut 4, ajoute 3 si la variable est inférieure à 2, et sinon retire 2.

```
a = 4
if a == 4:
    a = a + 2
elif a < 2:
    a = a + 3
else:
    a = a - 2
print(a)
```

Boucle

De manière commune, on peut utiliser les boucles **Tant Que** et **Pour**. Pour ce premier, on utilise le mot-clé **while** suivi d'une expression conditionnelle, ainsi que **:** pour commencer le bloc.

Par exemple, voici une boucle réalisant la somme 1+2+...+10.

```
s = 0
i = 1

while i < 11:
    s = s + i
    i = i + 1

print(s)
```

Boucle

Dans le cas du **for**, on précise un index qui sera dans un intervalle (**range**), suivi d'un **:** pour commencer le bloc.

Par exemple, voici une boucle réalisant elle-aussi la somme 1+2+...+10.

```
s = 0

for i in range(1, 11):
    s = s + i

print(s)
```

Fonction

Pour décarer une fonction, il faut utiliser le mot-clé **def** et définir le nom et les paramètres de la fonction. Tout comme la condition, le bloc de fonction commencera par **:** et son sein sera délimité par et grâce à l'indentation.

Par exemple, voici une fonction renvoyant le double d'un paramètre, ainsi que son appel.

```
def double(n):
    return n*2

a = double(4)
```

Remarque Vu que le typage est faible, on ne sait pas quel est le type de *n* fourni à la fonction.

Fonction

Vu que le typage est faible, on peut fournir n'importe quel type à la fonction. On peut toutefois préciser quels types on souhaite pour chaque paramètre, ainsi que le type de retour.
Cependant, bien que cela permet d'être plus clair et lisible, **cela ne change pas le fonctionnement de la fonction**. On peut toujours lui fournir des types autres que ceux écrits.

Si on reprend notre fonction précédente.

```
def double(n:int) -> int:
    return n*2

a = double(4)
```

Ici, on notifie que la fonction **double** prend des entiers en entrée et retourne des entiers.

TP

Objectif du TP :

- Créer une fonction qui prend une liste d'entiers positifs en entrée et retourne l'entier le plus grand de la liste.

Indication Indiquer le type de sortie ne suffit pas puisque ce n'est qu'une indication, il faut donc réaliser des vérifications avec la fonction **type**, en retournant un code erreur (par exemple **-1**) s'il y a un souci.