IngeSUP - Cours 01 - Opérateurs, variables et entrées sorties

Sommaire

Objectifs

- Présenter le langage Python ;
- Introduire les expressions, les opérateurs basiques et leurs priorités ;
- Comprendre ce qu'est une variable, son affectation et les types de variables que l'on retrouve ;
- Appréhender les fonctions de base qui permette une communication avec l'utilisateur d'un programme.

Supports Vidéo

Les supports vidéos suivants traitent des mêmes notions sur la plateforme ionisX :

- Module 1 : Python – introduction (https://courses.ionisx.com/courses/ref/m205/x/courseware/6c4b9c9a9b7f40f5a9d6304b59ddf6cc/)
- Module 2 : Types et Opérateurs (https://courses.ionisx.com/courses/ref/m61/x/course/)
- Module 3 : Variables et affectation en python (https://courses.ionisx.com/courses/ref/m203/x/course/)
- Module 4 : Les entrées/sorties - Clavier / Affichage (https://courses.ionisx.com/courses/ref/m188/x/course/)

Définitions

Algorithme

A l'ère du Big Data, du Machine Learning et de l'intelligence artificielle, les algorithmes sont devenus omniprésents dans nos vies. Mais en fait, qu'est-ce qu'un algorithme ?

Le terme algorithme fut inventé par le mathématicien Mohammed Ibn Musa-Al Khwarizmi, dans le courant du neuvième siècle avant Jésus Christ. Avec l'essor de l'intelligence artificielle, ce terme est de plus en plus utilisé.

Apprendre à construire des algorithmes et à évaluer leurs capacités, c'est le but de cette matière.

Dans le domaine des mathématiques, dont le terme est originaire, un algorithme peut être considéré comme un **ensemble d'opérations ordonné et fini devant être suivi dans l'ordre pour résoudre un problème**.

En guise d'exemple très simple, prenons une recette de cuisine. Dans chaque recette, une procédure spécifique doit être suivie dans l'ordre. Les différentes étapes de la recette représentent les opérations qui constituent l'algorithme.

Langage de programmation

Pour effectuer une tâche, quelle qu'elle soit, un ordinateur a besoin d'un programme informatique. Or, pour fonctionner, un programme informatique doit indiquer à l'ordinateur ce qu'il doit faire avec précision, étape par étape.

L'ordinateur exécute ensuite le programme , en suivant chaque étape de façon mécanique pour atteindre l'objectif. Or, il faut aussi dire à l'ordinateur comment faire ce qu'il doit faire.

C'est le rôle de l'algorithme informatique mais aussi des **langages de programmation** qui transforment les algorithmes en instructions compréhensibles par les ordinateurs.

Le langage Python

Python est un langage de programmation **moderne**, **généraliste**, **orienté objet** et de **haut niveau**.

Voici ses caractéristiques techniques principales:

- **typage dynamique** : Pas besoin de définir le type des variables, des arguments ou des types de retour des fonctions, il est inféré à l'exécution.

- **gestion automatique de la mémoire** : Pas besoin d'allouer ou de désallouer explicitement pour les variables et les tableaux de données. Pas de problème de fuite de mémoire.

- **interprété** : Pas besoin de compiler le code. L'interpréteur Python lit et exécute directement le code python.

Python étant un langage de programmation interprété et typé dynamiquement, l'exécution du code python peut être lente comparée à des langages de programmation compilés et typés statiquement tels que le C ou le Fortran.

> 🧠 **Culture**
>
> Le langage Python a été crée en 1971 par **Guido van Rossum** qui travaillait à cette époque à l'institut néerlandais public de recherche en informatique. Le nom du langage est un hommage aux Monthy python, un groupe humoriste anglais ayant notamment réalisé une série télévisée et quatre long métrages.

> En 2001 la *python software foundation* est créée, c'est une association à but non lucratif qui a pour but de développer le langage. Celui-ci est distribué sous licence libre spécifique, la Python licence.

Evaluation des expressions

L'évaluation des expressions (et plus tard des variables) est une étape incontournable de la programmation.

Les opérateurs simples

Une cellule de code Python peut-être utilisée comme une calculatrice. Si l'on considère l'expression $5 + 9$, nous pouvons calculer le résultat en tapant:

Entrée [1]:

```
5+9
```

Out[1]:

```
14
```

On peut utiliser la notation scientifique pour exprimer les valeurs. Par exemple, le nombre 8×10^{-2} peut être entrée comme `0.08` or `8e-2` . Ceci peut-être facilement vérifié à partir du calcul de l'expression :

Entrée [ ]:

```
0.08 - 8e-2
```

Nous pouvons aussi avoir recours aux puissances d'un nombre. Le calcul de 3^4 sera par exemple :

Entrée [ ]:

```
3**4
```

Le reste d'une division est obtenu à partir de l'opérateur *modulo* `%` :

Entrée [ ]:

```
11 % 3
```

Pour obtenir le quotient, nous utiliserons la *division entière*, symboliséen par `//` :

Entrée [1]:

```
11 // 3
```

Out[1]:

```
3
```

En résumé, python peut faire des additions des soustractions, des multiplications et des divisions. Pour ce faire, il manipule les mêmes opérateurs que la calculatrice `+` , `-` , `*` , `/` .

Les priorités sont les mêmes qu'en mathématiques.

Priorité des opérateurs

L'ordre de priorités dans les opérations fait référence à l'ordre dans lequel les opérations sont effectuées, par exemple la multiplication avant l'addition.

La plupart des langages de programmation (dont Python) suivent les règles mathématiques basiques en matière de priorités des opérateurs.

Considérons l'expression $4 \cdot (7 - 2) = 20$. Sans faire attention, nous pourrions écrire :

Entrée [ ]:

```
4*7 - 2
```

Dans le calcul ci-dessus, `4*7` est évalué en premier, puis `2` est soustrait car la multiplication (`*`) vient avant la soustraction (`-`) en termes de priorité. **Nous pouvons contrôler l'ordre des opérations en utilisant les parenthèses** :

Entrée [2]:

```
4*(7 - 2)
```

Out[2]:

```
20
```

Un autre exemple classique peut être décrit ci -dessous :

$$\frac{10}{2 \times 50} = 0.1$$

Le code suivant...

Entrée [ ]:

```
10/2*50
```

Ne traduit pas l'expression correctement !

A la place on écrit :

Entrée [ ]:

```
10/(2*50)
```

Variables et affectation

Les variables

En programmation, on va pouvoir accéder facilement aux **emplacements de la mémoire qui stockent des valeurs** en leur donnant un nom, c'est ce que l'on appelle une **variable**.

Le choix du nom pour nos variables est libre en Python. Il faut cependant **respecter les règles usuelles suivantes**.

- Le nom doit commencer par une lettre ou par un underscore ;
- Le nom d'une variable ne doit contenir que des caractères alphanumériques courants (pas d'espace dans le nom d'une variable ni de **caractères spéciaux** comme des caractères accentués ou tout autre signe) ;
- On ne peut pas utiliser certains mots qui possèdent déjà une signification spéciale pour le langage, on parle de **mots réservés** (*range* , *print* , *input* ...). Les mots réservés s'appellent aussi **les mots-clés**.

Les suite des caractères (non entourés par des guillemets) qui satisfont à ces principes s'appellent des **identifiants**.

L'affectation

L'affectation consiste à donner une valeur à une variable.

> 📝 **Note**
>
> En Python **l'affectation** s'utilise à l'aide du symbole **(=)**
>
> A ne pas confondre avec l'égalité ! (qui se note ==)

Voici quelques exemples :

Entrée [ ]:

```
a = 2    # On met 2 dans la variable a
m = 5    # On met 5 dans la variable m
a = m + a
a = a + 1
m = a * m
m = 2 * m
```

Voici un tableau de suivi des variables `a` et `m`, remplissez les valeurs manquantes

Ligne	a	m
1	2	.
2	2	5
3	7	5
4	8	?
5	8	?
6	?	?

> 📝 **Note**
>
> Remarquons que quand on réalise une affectation la machine calcule le membre de droite et **l'affecte** à la variable de gauche.

Types des variables

Les données sont divisées selon leur nature en différents ensembles appelés **types**.

Les types les plus importants sont les **nombres entiers** *(type int)* , les **nombres décimaux** *(type float)* , **les chaînes de caractères** *(type str)* et le type qui regroupe des **valeurs de vérité** *(type bool)*.

Nous pouvons manipuler le type d'une variable à partir des instructions suivantes :

Entrée [ ]:

```
# Entier
n = 5
```

Entrée [ ]:

```
# Décimal
x = 1.0
```

Entrée [ ]:

```
# Booléen
c = True  # Attention à la majuscule !
```

> **Note**
>
> **Les nombres à virgules s'écrivent avec des points (.) en python**

Entrée [ ]:

```
# Chaine de caractère
s = "1.0"  # s = '1.0', fonctionne également
```

> **Note**
>
> En python, **les chaines de caractères** sont des suites de caractères entourés par des guillemets simples ' ou doubles " .
>
> > Par exemple on peut écrire "abracadabra" ou 'abracadabra' c'est la même chose

On peut parfois transformer une donnée d'un certain type en une donnée d'un autre type. Cela s'appelle le **transtypage** (ou cast en anglais).

Entrée [ ]:

```
a = 1

a = float(a)  # Cette opération convertie la variable entière 'a' en une variable réelle

print(a)
```

Entrée [ ]:

```
var = "12"

var = int(var)  # Cette opération convertie la chaine de caractère 'var' en une variable er

print(var)
```

> **Note**
>
> Le symbole **(+)** en python sert aussi à coller (on dit **concaténer** des chaînes de caractères)

Entrée [ ]:

```
"ab"+"cd"    # Devient "abcd"
```

> **Attention**
>
> On a vu que l'opérateur **(+)** sert aussi bien à **additionner des entiers** qu'à **coller deux chaînes de caractères entre elles**. Mais que se passe t-il lorsqu'on essaie de l'utiliser entre un entier et une chaîne de caractères ?
>
> **Réponse:** Une erreur ! 😮 (voir ci-dessous)

Entrée [ ]:

```
"hello"+5
```

> **Astuce**
>
> L'une des techniques ça peut être de transformer l'entier concerné en chaîne de caractères à l'aide de la fonction prédéfinie str
>
> > En python la fonction str transforme le nombre qu'on lui donne en argument en chaine de caractères représentant ce nombre. Ainsi **str(5)** va renvoyer **"5"**

Entrée [ ]:

```
# Du coup on a plus de problèmes pour coller 5 et hello
"hello"+str(5)
```

Entrées/Sorties utilisateur et quelques fonctions prédéfinies

Une **fonction** sert à donner un nom à un **ensemble d'instructions effectuant une tâche précise**.

A chaque fois que nous devons réaliser cette tâche, nous pouvons alors écrire le nom de la fonction plutôt que d'écrire l'ensemble d'instructions correspondant.

On dit qu'on appelle la fonction.

Appeler une fonction c'est écrire son nom en mettant ses paramètres entres parenthèses.

> 📝 **Note**
>
> Comme tout langage de programmation, Python possède une multitude de fonctions déjà définies. Nous allons en voir quelques-unes

1. La fonction len()

La fonction `len()` a un paramètre de type **chaine de caractères**. Elle renvoie la longueur de la chaîne.

Entrée [ ]:

```
len('hello')
```

Entrée [ ]:

```
len('bonjour')
```

La fonction `len()` produit un résultat : **l'entier** correspondant au **nombre de caractères de la chaîne d'entrée**. On peut donc combiner un appel de fonction avec d'autres expressions pour faire des calculs...

Entrée [ ]:

```
len('bonjour') + 4
```

Entrée [ ]:

```
len('bon')+len('jou')
```

2. La fonction print()

Cette fonction permet d'afficher n'importe quelle valeur fournie en paramètre (c'est-à-dire entre parenthèses).

Entrée [ ]:

```
print('bonjour')
print(221)
print(len('bonjour'))
```

On peut l'utiliser en mettant une suite de valeurs séparées par des virgules dans les parenthèses.

Entrée [ ]:

```
print('bleu', 'blanc', 12)
```

On peut mélanger des valeurs stockées dans des variables avec des valeurs mises directement entre guillemets en séparant le tout par des virgules.

Entrée [ ]:

```
a=12
b=7
print(a,"nains et",b,"lutins marchent dans la forêt")
```

Il peut être intéressant dans la présentation des résultats d'empêcher le passage à la ligne. On utilise pour cela la commande `end=' '`

Entrée [ ]:

```
print("premier résultat", end =" - ")
#print("premier résultat")
print("second résultat")
```

A noter qu'une écriture équivalente serait :

Entrée [ ]:

```
print("premier résultat"," - ","second résultat")
```

Il peut également être intéressant de passer volontairement à la ligne. On utilise pour cela la commande `\n`

Entrée [ ]:

```
print("ceci est le début d'une phrase \nceci est la suite à la ligne")
```

Enfin, il peut être utlise d'utiliser des séparateurs. On utilise pour cela la commande `sep` :

Entrée [ ]:

```
print("premier résultat","deuxième résultat","troisième résultat",".",sep = " -- ")
```

> **Astuce**
>
> En programmation il existe des techniques pour s'assurer que les caractères ne soient pas **interprétés**, on appelle ce procédé **l'échappement de caractères** Dans un print on utilise le caractère \ .
>
> On s'asssure ainsi que les caractères soient traités pour ce qu'ils sont et non pas interprétés ; par exemple si on veut les afficher "tels quels".

Ici les guillemets s'afficheront "tels quels" et ne seront pas vus comme des délimiteurs.

Entrée [ ]:

```
print("le mot \"Pierre\" est entre guillements")
```

3. La fonction input()

Elle permet **d'inviter l'utilisateur à entrer une chaîne de caractères**. L'éxécution de cette fonction provoque l'arrêt de l'exécution du programme. L'utilisateur doit entrer une suite de caractères au clavier **puis appuyer sur *entrer***.

La chaîne founie est alors le résultat de la fonction et l'exécution du programme se poursuit.

Entrée [ ]:

```
print('Entrez votre nom')
reponse=input()
print('Bonjour', reponse, '!')
```

On peut mettre une invite **entre les parenthèses du input()** et **entre les guillemets**. Celle-ci sera affichée pour indiquer à l'utilisateur le genre de valeur qu'on veut qu'il rentre.

Entrée [ ]:

```
reponse=input('Entrez votre nom:')
print('Bonjour',reponse,'!')
```

Entrée [ ]:

```
n = input ("entrer un nombre entier : ")
print("Le nombre que vous venez d'entrer est: ", n)
```

> 🚧 **Attention**
>
> Attention, la fontion **input() retourne toujours une chaîne de caractères**, même si l'utilisateur a entré un nombre. Si vous voulez récupérer la valeur entrée sous forme de nombre, il faudra la convertir en utilisant int(valeur) ou float(valeur)

Entrée [ ]:

```
reponseInt = int(input('Entrez un nombre: '))
print("Le double de votre nombre est :", reponseInt*2)
```

4. La fonction type()

La fonction `type()` donne le type de la valeur donnée en paramètres.

Entrée [ ]:

```
# Booléen
c = True  # Attention à la majuscule !
type(c)
```

Entrée [ ]:

```
# Entier
n = 5
type(n)
```

Entrée [ ]:

```
# Décimal
x = 1.0
type(x)
```

Entrée [ ]:

```
# Chaine de caractère
s = "1.0"  # s = '1.0', fonctionne également
type(s)
```

On peut d'ailleurs voir en interne le fonctionnement de la fonction `input()` décrite précédemment.

Entrée [ ]:

```
v = input("Entrez une valeur")
type(v)
```

Entrée [ ]:

```
n = int(input("entrer un nombre entier : "))
type(n)
```

Entrée [ ]:

```
r = float(input("entrer un réel : "))
type(r)
```

Exercices de TD

Vous pouvez maintenant vous exercer à partir du notebook [TD 01 - Opérateurs, variables et IO utilisateur (../TD/TD%2001%20-%20Op%C3%A9rateurs%2C%20variables%20et%20IO%20utilisateur.ipynb)](../TD/TD%2001%20-%20Op%C3%A9rateurs%2C%20variables%20et%20IO%20utilisateur.ipynb).