Informatique et Calcul Scientifique

Cours 1 : Introduction et variables

26.02.2025

Objectifs de la leçon

Le but de la leçon d'aujourd'hui est de :

- Faire connaissance et présenter le déroulement du cours
- Comprendre l'importance de l'ICS dans le contexte actuel
- Introduire les notions de type, de variable et d'affectation
- Les fonctions `print()` et `input()`
- Ecrire son premier programme

Point admin

Toutes les informations importantes se trouvent sur le Moodle du cours

- Le cours est enregistré et sera mis en ligne en fin de journée sur la chaîne mediaspace dédiée
- Les exercices auront lieu les mercredi de 17h15 à 19h00 en RLC E1 240, CM1 121, INJ 218 et INF 3, où 10 assistant.e.s seront présent.e.s pour vous encadrer
- Ils prendront la forme de travaux pratiques sur ordinateur et sont accessibles sur la plateforme Noto de l'Epfl
- N'hésitez pas à poser vos questions sur le forum du cours

Point admin

Répartition dans les salles d'exercices :

RLC E1 240	SV, PH, MT, GC, EL, MA
CM1 121	IN
INJ 218	GM, AR, MX
INF 3	SIE, SC, CGC + celles et ceux sans ordinateur

Plan du cours

Ce semestre est séparé en deux parties principales de 7 semaines chacune environ :

- Introduction au langage Python
  - Ecriture de petits programmes
  - Notions d'algorithmique
  - Librairies scientifiques
- Elements d'analyse numérique
  - Le côté pratique de l'Analyse

Aucun prérequis n'est demandé !

But du cours

- Appréhender les différents outils numériques et en comprendre l'importance
- Etre capable d'écrire un petit programme pour résoudre une situation décrite
- Pouvoir lire et comprendre un code avancé dans n'importe quel langage
- Utiliser son savoir faire en programmation pour analyser des problèmes mathématiques ou physiques

Qu'est-ce que l'informatique ?

Définition de l'informatique :

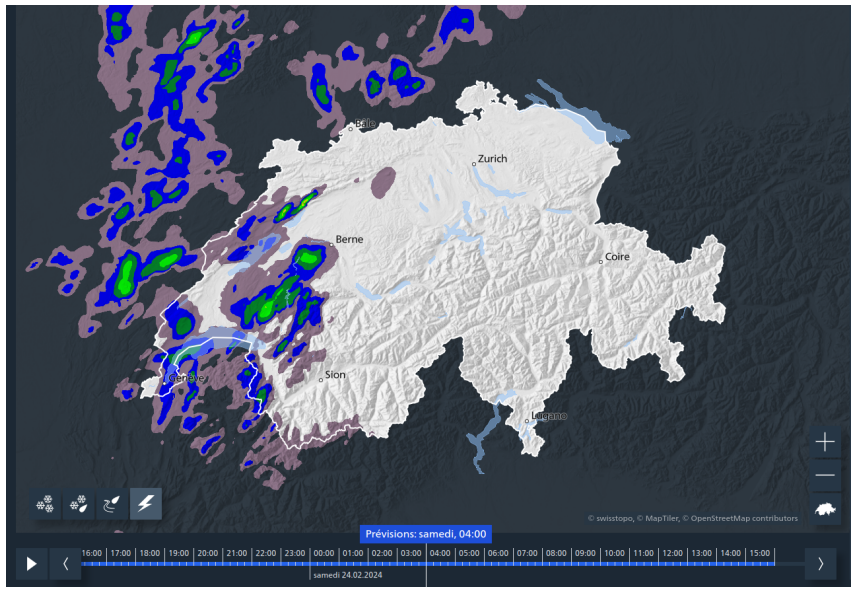
Théorie et traitement de l'**information** à l'aide de **programmes** mis en œuvre sur **ordinateurs**. (*Le Robert, 2024*)

- En informatique, l'information est représentée par un ensemble de **données**.
- Un ordinateur est **automate programmé** effectuant des opérations sans intervention humaine sur des données.
- Un programme est une **méthode de communication** avec la machine pour lui demander d'effectuer ces tâches.

Domaines d'application de l'informatique

Les outils informatiques assistent l'humain pour des tâches de :

- Calcul scientifique
  Simulation, modélisation, ...
- Automatisation
  Tâches répétitives, ...
- Gestion de l'information
  Stockage, analyse de données, ...

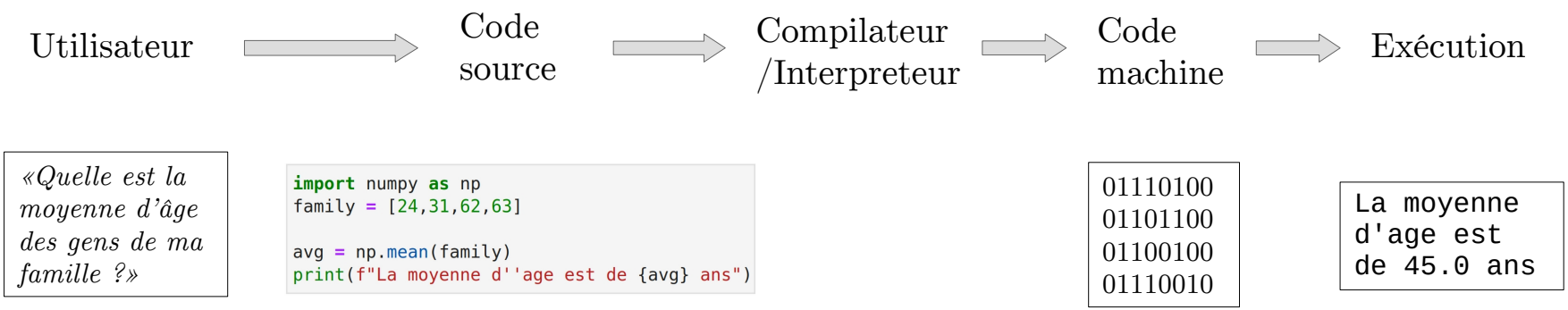


Dans tous ces domaines, l'ingénieur.e implémentera des algorithmes [1] sous forme de programmes, qui seront exécutés par un (ou plusieurs) ordinateur(s).

1. Un algorithme est une procédure pour résoudre un problème alors qu'un programme est une implémentation spécifique d'un algorithme.

Communication avec l'ordinateur

- L'ordinateur parle le **langage machine**. Il s'agit d'une suite de bits (1 et 0) qui est incompréhensible pour un être humain.
- Pour communiquer avec la machine, on utilise un **langage de programmation**. Comme tout langage, il possède un vocabulaire et une syntaxe propre ainsi qu'une sémantique associée.



Le langage Python 

Le langage que nous allons utiliser dans le cadre de ce cours est le **langage Python**.

- Il s'agit du langage le plus populaire au monde [2] [3], très répandu dans le domaine de l'automatisation et de la gestion des données (machine learning entre autres).
- Il possède une syntaxe intuitive, et une grande puissance de calcul malgré une vitesse limitée.
- La présence d'une multitude de librairies et modules en ligne simplifie l'écriture de programmes complexes.
- Sa communauté très active sur les forums (stackoverflow).
- Il s'agit d'un langage évolutif (2 versions par an),
- .. et surtout, il est gratuit !

2. https ://www.tiobe.com/tiobe-index/
3. https ://spectrum.ieee.org/the-top-programming-languages-2024

Interface utilisateur – ordinateur

Avant de se lancer dans la programmation, il faut s'assurer que tous les outils nécessaires sont installés sur son poste de travail :

- une **distribution** Python.
  Il s'agit de l'ensemble de fichiers nécessaires à l'interprétation d'un code en Python. Deux distributions sont communément utilisées : cPython et Anaconda
- un **environnement de développement intégré** (IDE).
  Il s'agit d'un ensemble d'outils visant à faciliter le travail des programmeurs (éditeur de texte, débuggeur, compilateur rapide). Les plus répandus sont Jupyter Notebook, PyCharm, Visual Studio Code ou encore Spyder

Dans ce cours, nous allons utiliser un IDE Jupyter Notebook, configuré spécialement pour les besoins de l'EPFL [4].

4. https://noto.epfl.ch

Exécuter du code en ligne de commande

- Le moyen le plus simple d'exécuter du `Python` est de passer par un terminal. On peut lancer le *shell* (interpréteur de commandes) Python en y tapant la commande `python3`.
- Chaque instruction, ou **ligne logique**, est lue, exécutée et imprimée l'une après l'autre par l'interpréteur

```
>>> 1+1
2
>>> 1+ \
... 2 \
... +3
6
>>> print("Bienvenue au cours d'ICS")
Bienvenue au cours d'ICS
>>> # Ceci est un commentaire
>>>
```

Ecriture d'un ensemble d'instructions

- Lorsque les instructions que nous souhaitons transmettre à la machine sont plus complexes, on écrit généralement un programme, ou **script**. Il s'agit d'un ensemble d'instructions écrites dans un fichier texte avec l'extension `.py`

```
1+1
1+ \
2\
+3
print("Bienvenue au cours d'ICS")
#Ceci est un commentaire
```

- Le script doit alors être compilé avant de pouvoir être exécuté. Pour des projets simples, et dans la plupart des environnements de programmation, ceci est fait de manière automatique.

```
Bienvenue au cours d'ICS
```

Types

- Un programme Python manipule des **objets** qui ont des **valeurs**. Chaque objet a un **type** qui détermine les opérations qu'on peut effectuer sur cet objet.
- Dans le shell Python :

```
>>> type(2)
<class 'int'>
>>> type(2.0)
<class 'float'>
>>> type(2+0j)
<class 'complex'>
>>> type("2")
<class 'str'>
```

Les types numériques

Ils sont au nombre de trois :

- `int` est la classe des entiers
- `float` est la classe des réels (*nombres à virgule flottante*)
- `complex` est la classe des complexes

Opérations sur les `int` et les `float`

- On peut effectuer les opérations arithmétiques usuelles sur les `int` et les `float` : addition (+), soustraction (-), multiplication (*), division (/).
  Un `int` sera considéré comme un `float` au besoin.

```
>>> (4+3)*10 - 2
68
>>> 4 + 2.0
6.0
```

- L'opérateur / dénote la division de réels, qui produit un `float` même à partir de deux `int` :

```
>>> 8/5
1.6
>>> 8/2
4.0
```

Opérations sur les `int` et les `float`

- L'opérateur `**` calcule des puissances réelles :

```
>>> 2 ** 3
8
>>> 2.2 ** 3
10.648000000000003
>>> # (l'opération ci-dessous est bien
>>> # définie mathématiquement)
>>> 2.2 ** 3.1
11.521534126785717
```

- Les opérateurs `//` et `%` calculent respectivement le quotient et le reste de la division entière (voir série).

Le type `str`

- Une chaîne de caractères (de type `str`) est une suite de caractères (alphanumériques, caractères spéciaux) entourés de double quotes (`‘‘abc’’`) ou de simples quotes (`’abc’`).

```
>>> type("abc")
<class 'str'>
>>> type('abc')
<class 'str'>
```

- La fonction `len()` donne la longueur d'une chaîne de caractères :

```
>>> len("abc")
3
>>> len('ab cd')
5
```

Opérations sur les str

Les opérateurs `+` et `*` s'appliquent aussi à des `str` mais ils n'ont pas la même signification que pour des nombres !

- `+` effectue la concaténation de deux chaînes de caractères
- `*` effectue la concaténation d'une chaîne de caractères avec elle-même un nombre donné de fois.

```
>>> "abc" + "def"
'abcdef'
>>> "une" + " " + "phrase"
'une phrase'
>>> 'ha' * 3
'hahaha'
>>> 3 * 'hi'
'hihihi'
>>> "3 * hi"
'3 * hi'
```

Variables et affectations

- Dans un programme, on a souvent besoin de stocker une valeur pour la réutiliser plus loin. Pour ce faire, on utilisera des **variables**.
- Une variable est une **référence** à un objet en mémoire, un **nom** par lequel on désigne cet objet.
- Une variable est créée par une instruction d'**affectation**, de la forme `nom = expression`.
- Lorsqu'une expression fait intervenir une variable (par exemple `x+2` ci-dessous), la valeur de l'objet référencé par la variable est utilisée pour calculer la valeur de l'expression.

```
>>> x = 3
>>> x
3
>>> x + 2
5
```

Nommer une variable

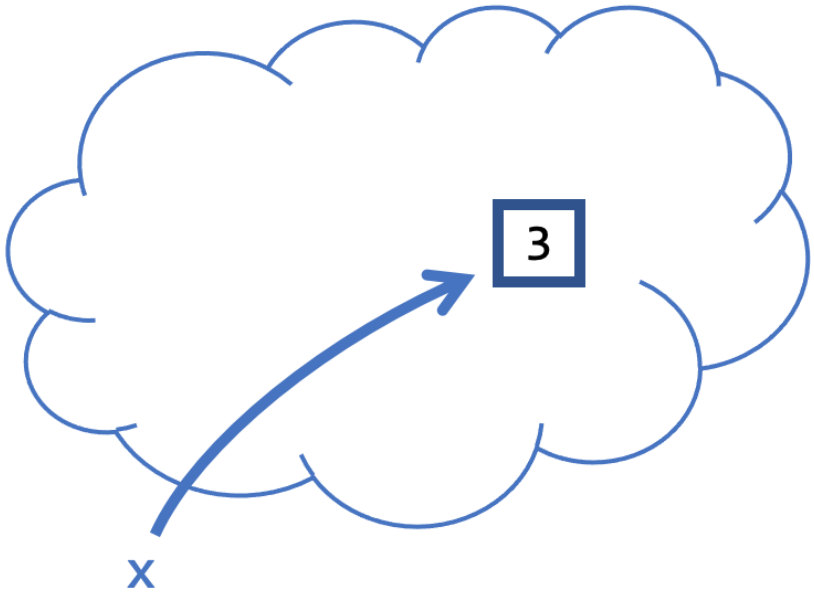
On peut nommer une variable comme on veut, à condition

- de ne pas utiliser un des mots-clés réservés du langage Python, comme `class`, `def`, `lambda`... : ce sont des mots qui ont une signification et une fonction précise dans le langage Python (pour une liste complète voir ici.)
- que le nom de variable contienne uniquement des lettres, chiffres, et underscore, et commence par une lettre ou un underscore
- de se rappeler que les noms de variable en Python sont sensibles à la casse.

Mais même si on peut nommer une variable comme on veut, il vaut mieux utiliser des noms parlants, et suivre les conventions de nommage de variables.

Affectation de variable

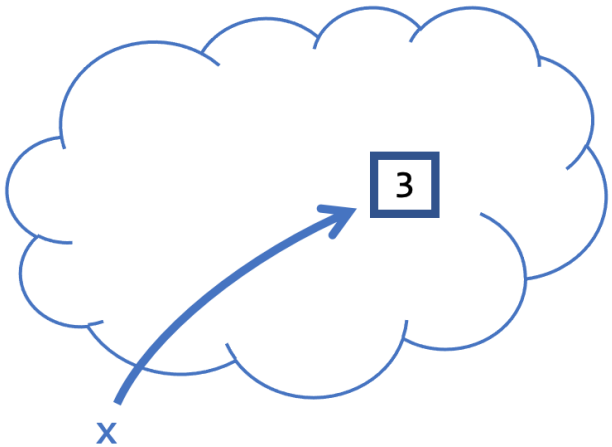
Lors de l'exécution de l'instruction `x = 3`, un entier de valeur 3 est stocké en mémoire, et `x` va pointer vers l'adresse en mémoire où cet objet est stocké.



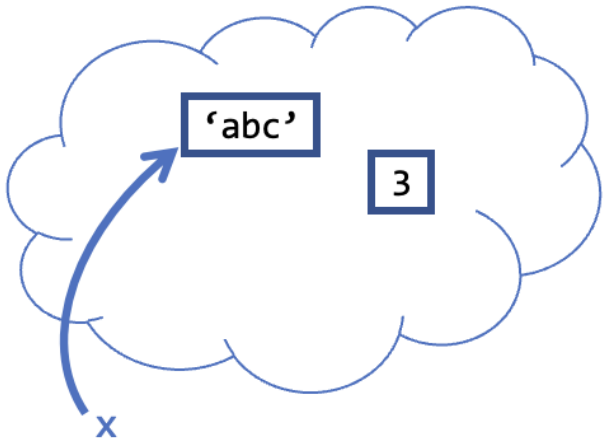
Affectation de variable

Une variable peut être **réaffectée** :

```
>>> x = 3
>>> x * 2
6
>>> x ='abc'
>>> x * 2
'abcabc'
```



affectation `x = 3`

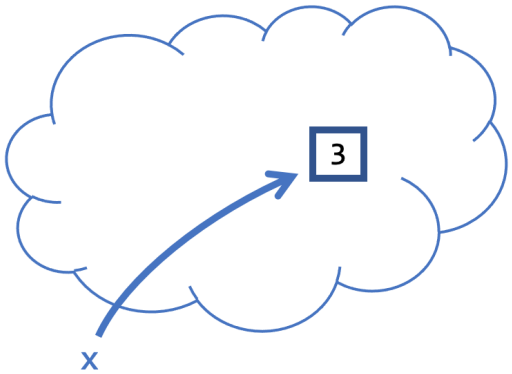


réaffectation `x = 'abc'`

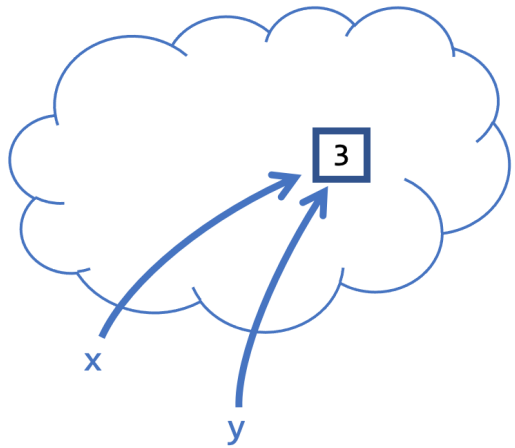
Affectation de variable

L'expression affectée à une variable peut dépendre d'une autre variable :

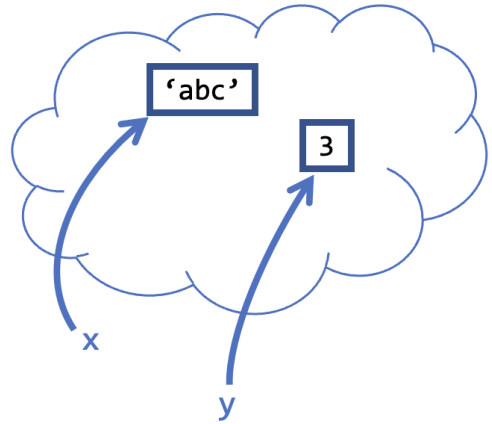
```
>>> x = 3
>>> y = x
>>> x = 'abc'
>>> x
'abc'
>>> y
3
```



affectation `x = 3`



affectation `y = x`

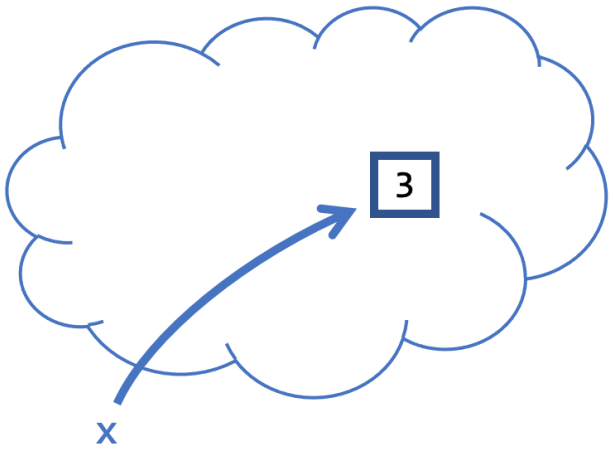


réaffectation
`x = 'abc'`

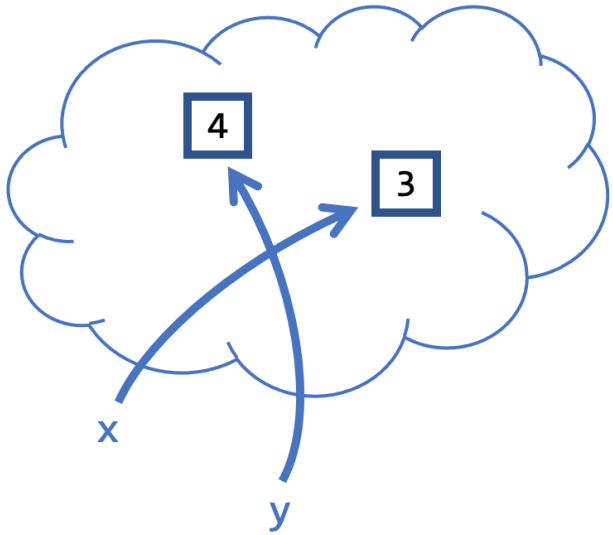
Affectation de variable

L'expression affectée à une variable peut dépendre d'une autre variable ...

```
>>> x = 3
>>> y = x + 1
>>> x
3
>>> y
4
```



affectation `x = 3`

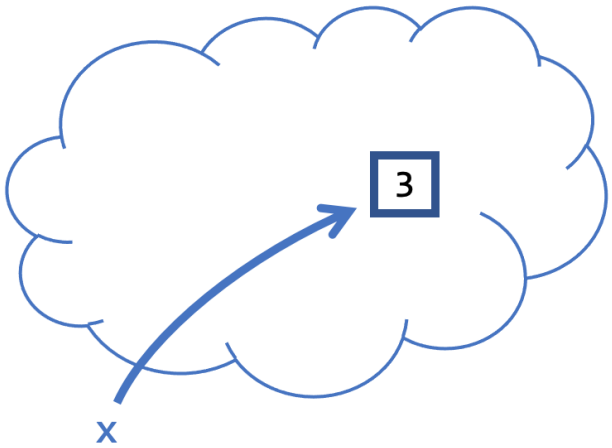


affectation `y = x+1`

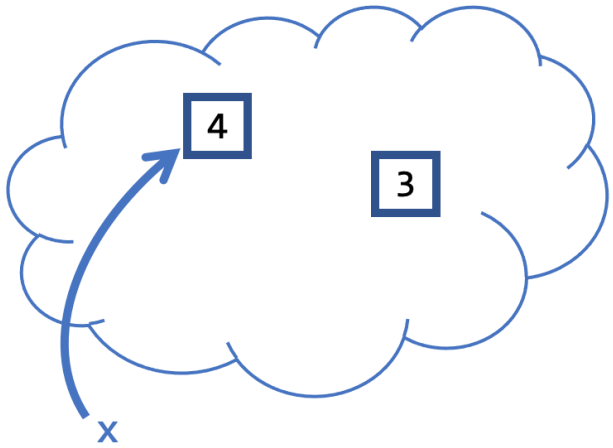
Affectation de variable

... ou de la variable elle-même :

```
>>> x = 3
>>> x = x + 1
>>> x
4
```



affectation `x = 3`



réaffectation `x = x+1`

Affectation de variable

- Le signe `=` dans une affectation de variable n'a rien à voir avec l'égalité au sens mathématique !
- Dans certains langages de programmation, l'opérateur d'affectation est `<-` .
- On peut se représenter l'affectation `x = 3` comme `x←3` : "on donne à `x` la valeur `3`".
- De même, on peut se représenter l'affectation `x = x+1` comme `x←x+1` :
  "on donne à `x` la valeur de l'expression de droite, qui est égale à la valeur actuelle de `x` à laquelle on ajoute `1`".

Affectation de variable

- On peut écrire l'instruction `x = x + 1` de manière équivalente mais plus concise : `x += 1`

```
>>> x = 3
>>> x += 1
>>> x
4
```

- On peut de même écrire de manière plus concise les instructions `x = x - 2`, `x = x * 4`...

```
>>> x = 3
>>> x -= 2
>>> x *= 4
>>> x /= 3
>>> x
1.3333333333333333
```

Valeur vs identifiant

Attention, deux objets qui ont la même valeur ne sont pas forcément le même objet !

```
>>> x = "I love Python"
>>> y = "I love Python"
>>> x
'I love Python'
>>> y
'I love Python'
>>> id(x)
4378525744
>>> id(y)
4378525872
```

- Lorsqu'un objet est créé en mémoire, un identifiant **unique** lui est associé. `id(x)` donne l'identifiant de l'objet vers lequel la variable `x` est une référence.

Valeur vs identifiant

- A comparer avec

```
>>> x = "I love Python"
>>> y = x
>>> id(x)
4378187952
>>> id(y)
4378187952
```

- A la différence de l'instruction

`y = "I love Python"`

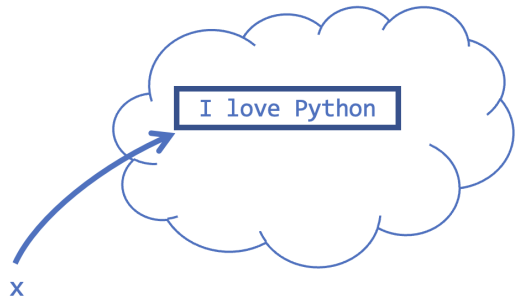
qui crée une nouvelle chaîne de caractère de valeur `"I love Python"`, l'instruction

`y = x`

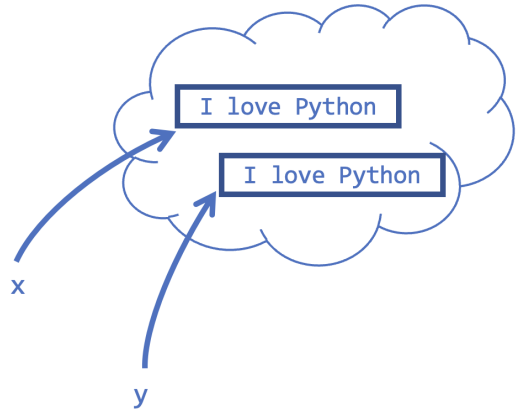
affecte à la variable `y` l'objet référencé par la variable `x`.

Valeur vs identifiant

- Dans le premier cas :

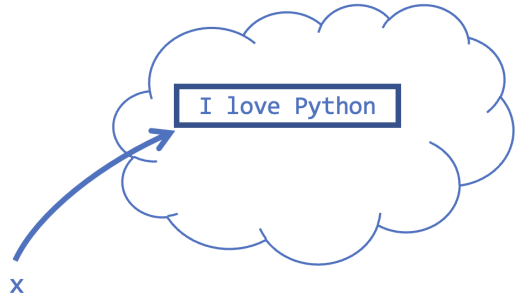



```
x = "I love Python"
```

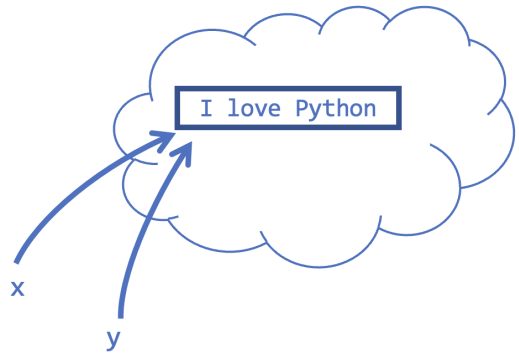



```
y = "I love Python"
```

- Dans le deuxième cas :




```
x = "I love Python"
```




```
y = x
```

`input()` et `print()` : entrée et sortie standard

Les fonctions `input()` et `print()` permettent à un programme Python d'interagir avec l'utilisateur :

- `input()` permet à un programme Python de récupérer une chaîne de caractères entrée par l'utilisateur au clavier (*l'entrée standard*)
- `print()` permet à un programme Python d'afficher une chaîne de caractères à l'écran (*la sortie standard*).

print() : sortie standard

- La fonction `print()` affiche à l'écran la valeur qu'on lui donne en argument, suivie d'un retour de ligne.
- Notre premier programme Python, constitué d'une seule instruction :

```
#ce programme produit l'affichage:
# Hello, world!

print("Hello, world!")
```

- Si on donne en argument à `print()` une instruction, le résultat de cette instruction sera affiché :

```
print(1 + 1)
x = 3
print(x*2+0.5)
```

Output :

```
2
6.5
```

print() : sortie standard

- Pour une variable `x` déjà définie, `print(x)` affiche la valeur de l'objet référencé par `x`. De même pour une expression faisant intervenir `x`, la valeur de l'objet référencé par `x` interviendra dans le calcul de l'expression.

```
x = 3
print(x)
print(x + 10)
```

Output :

```
3
13
```

- `print()` peut prendre en entrée plusieurs *arguments* séparés par des virgules. Elle affiche ces arguments dans l'ordre, séparés par des espaces.

```
x = 3
print("x = ", x, "\net x+10 =", x+10)
```

Output :

```
x = 3
et x+10 = 13
```

Chaînes de caractères : caractères d'échappement

Les caractères d'échappement sont précédés d'un backslash et permettent de contrôler l'affichage d'une chaîne de caractères :

- `\t` affiche un tab (espace blanc)
- `\n` affiche un retour de ligne
- `\'` et `\"` affichent une simple ou double quote

```
print("a\nb\tc")
print("c'est l'automne")
print('c\'est l\'automne')
print("\\ est un backslash")
```

```
Output :
a
b    c
c'est l'automne
c'est l'automne
\  est un backslash
```

Chaînes de caractères formatées : f-strings

- Si on précède une chaîne de caractères de la lettre `f` ou `F`, on peut y inclure entre accolades des expressions dépendant d'une variable, qui seront calculées au moment de l'affichage.

```
x = 3
print("x =", x, "et x + 10 =", x + 10)
print(f"x = {x} et x + 10 = {x + 10}")
```

```
Output :
x = 3, x + 10 = 13
x = 3, x + 10 = 13
```

- Si on fait suivre l'expression dans l'accolade d'un signe `=`, l'expression littérale sera affichée, suivie du signe `=` et de la valeur de l'expression.

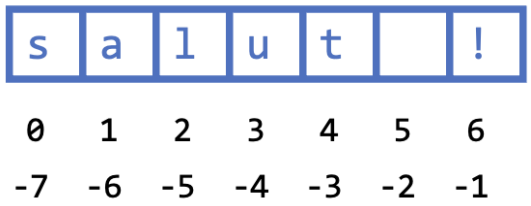
```
x = 3
print(f"{x = } et {x + 10 = }")
y = 5
print(f"on a {y - 2 = }")
```

```
Output :
x = 3 et x + 10 = 13
on a : y - 2 = 3
```

Chaînes de caractères : indexage

On peut **indexer** les caractères individuels d'une chaîne de caractères de longueur `n` :

- par `0`, `1`, ..., `n-1`
- mais aussi depuis la fin par `-1`, `-2`, ..., `-n`.




```
s = "salut !"
print(len(s))
print(s[0])
print(s[6])
print(s[-1])
print(s[-7])
```

```
Output :
7
s
!
!
s
```

Si on essaie d'accéder à un index `i` non entier de `s`, ou à `i` $\geq$ `len(s)`, ou `i` $<$ `- len(s)`, une exception est générée à l'exécution.

Chaînes de caractères : slicing

Il est possible de ne sélectionner qu'une partie d'une chaîne de caractères.

- Pour une chaîne de caractères `s`, `s[i:j]` est une sous-chaîne qui contient les éléments de `s` entre les indices `i` (inclus) et `j` (exclus).
- Si `i` est omis, il est pris à `0` par défaut. Si `j` est omis, il est pris à `len(s)` par défaut. `s[:]` est donc une copie de `s`.

```
s = "abcdef"
print(s[2:4])
print(s[:3], s[2:])
s2 = s[:]; print(s2)
```

Output :

```
cd
abc cdef
abcdef
```

Nous verrons plus tard qu'on peut définir du slicing sur d'autres types d'objets.

Chaînes de caractères : slicing

On peut également définir un pas pour le slicing : `s[i:j:k]` est un `string` contenant les éléments de `s` d'indices `i`, `i + k`, `i + 2k`, ... jusqu'à `j` exclus.

- Attention, le pas peut être négatif !
- Le pas est pris à 1 par défaut.

```
s = "abcdef"
print(s[0:7:2])
print(s[0:len(s):2], s[::2])
print(s[::-1])
```

Output :

```
ace
ace ace
fedcba
```

input() : entrée standard

La fonction `input()` permet à un programme Python de prendre en entrée une chaîne de caractères (`string`) entrée par l'utilisateur au clavier.

L'instruction `nom = input("Entrez votre nom s'il vous plaît")`

1. affiche la phrase `Entrez votre nom s'il vous plaît` à l'écran
2. suspend le déroulement du programme jusqu'à ce que l'utilisateur ait entré une chaîne de caractères et un retour de ligne
3. affecte cette chaîne de caractères à la variable `nom` .

Exécutez le code suivant :

```
nom = input("Entrez votre nom s'il vous plait")
print("bonjour", nom, "!")
print(type(nom))
```

`input()` : entrée standard

On désire écrire un programme qui demande à l'utilisateur d'entrer deux nombres, et qui affiche leur somme.

input() : entrée standard

On désire écrire un programme qui demande à l'utilisateur d'entrer deux nombres, et qui affiche leur somme.

- Quelle est l'erreur dans le programme suivant ?

```
x1 = input("Entrez le premier nombre: ")
x2 = input("Entrez le second nombre: ")
s = x1 + x2
print(f"la somme de ces deux nombres vaut {s}")
```

input() : entrée standard

On désire écrire un programme qui demande à l'utilisateur d'entrer deux nombres, et qui affiche leur somme.

- Quelle est l'erreur dans le programme suivant ?

```
x1 = input("Entrez le premier nombre: ")
x2 = input("Entrez le second nombre: ")
s = x1 + x2
print(f"la somme de ces deux nombres vaut {s}")
```

- Indice : Quel est le type de `x1` et `x2` ?

input() : entrée standard

On désire écrire un programme qui demande à l'utilisateur d'entrer deux nombres, et qui affiche leur somme.

- Quelle est l'erreur dans le programme suivant ?

```
x1 = input("Entrez le premier nombre: ")
x2 = input("Entrez le second nombre: ")
s = x1 + x2
print(f"la somme de ces deux nombres vaut {s}")
```

- Indice : Quel est le type de `x1` et `x2` ?
- `x1` et `x2` sont de type `str` et `s` correspondra à la **concaténation** de ces deux chaînes de caractères.

Casting

Le **casting** nous permet de *convertir* un objet d'un certain type en un objet d'un autre type (en fait, un nouvel objet est créé).

- La fonction `int()` crée, si possible, un objet de type `int`, à partir d'un objet de type `float` (en tronquant sa partie décimale) ou `str`.

```
a = int(2.7)
print(f"{a = }, {type(a) = }")
b = int(-2.7)
print(f"{b = }, {type(b) = }")
c = int("-27")
print(f"{c = }, {type(c) = }")
```

Output :

```
a=2, type(a) = <class 'int'>
b=-2, type(b) = <class 'int'>
c=-27, type(c) = <class 'int'>
```

- Mais les instructions `a = int("abc")` ou `a = int("2.7")` produiront une erreur.

Casting

De la même manière :

- La fonction `float()` crée, si possible, un objet de type `float`, à partir d'un objet de type `int` ou `str`.

```
print(float("2.7"))
print(float(2))
```

Output :

```
2.7
2.0
```

- La fonction `str()` crée un objet de type `str` à partir d'un objet de type numérique.

```
print(str(1.2))
print(2*str(1.2))
```

Output :

```
1.2
1.21.2
```

La somme de deux nombres

Revenons à notre problème : calculer la somme de deux nombres entrés par l'utilisateur.

- La fonction `input()` produit un objet de type `str` qui doit être converti en entier à l'aide de la fonction `int()`.
- Notre code devient :

```
x1 = int(input("Entrez le premier nombre:"))
x2 = int(input("Entrez le second nombre:"))
s = x1 + x2
print(f"la somme de ces deux nombres vaut {s}")
```

Points à retenir

- Une variable est un chemin d'accès vers une adresse en mémoire (`id`) où un objet est stocké.
- Lors de la création d'un objet, un `type` lui est automatiquement assigné.
- Les mêmes opérations sur des variables de types différents donneront des résultats différents.
- La fonction `print()` permet d'afficher la valeur d'un objet.
- La fonction `input()` permet à l'utilisateur de fournir un objet de type `str` à l'ordinateur au travers de son clavier.