Introduction à l'algorithmique et à la programmation

Cours 1

Anthony Labarre

Université Gustave Eiffel

Avant de commencer

Installez Python 3 (et pas 2) :

- installez le package `python3`
- téléchargez la dernière version (normalement : 3.8.3) sur `https://www.python.org/`

Attention : sous Windows, suivez le programme d'installation **mais ajoutez python à la variable PATH !**

Sans cela, vous ne pourrez pas taper `python` dans le terminal.

Aspects pratiques

Organisation :

- 10 séances de 2h de cours / TD
- 12 séances de 2h de TP

Evaluation :

- une note de TP : vous rendez chaque TP, tous seront notés
- un projet de programmation
- un examen

Ressources :

- `http://igm.univ-mlv.fr/~alabarre/teaching.php`
- une page sur l'intranet de l'UPEM (voir TPs)

Plan d'aujourd'hui

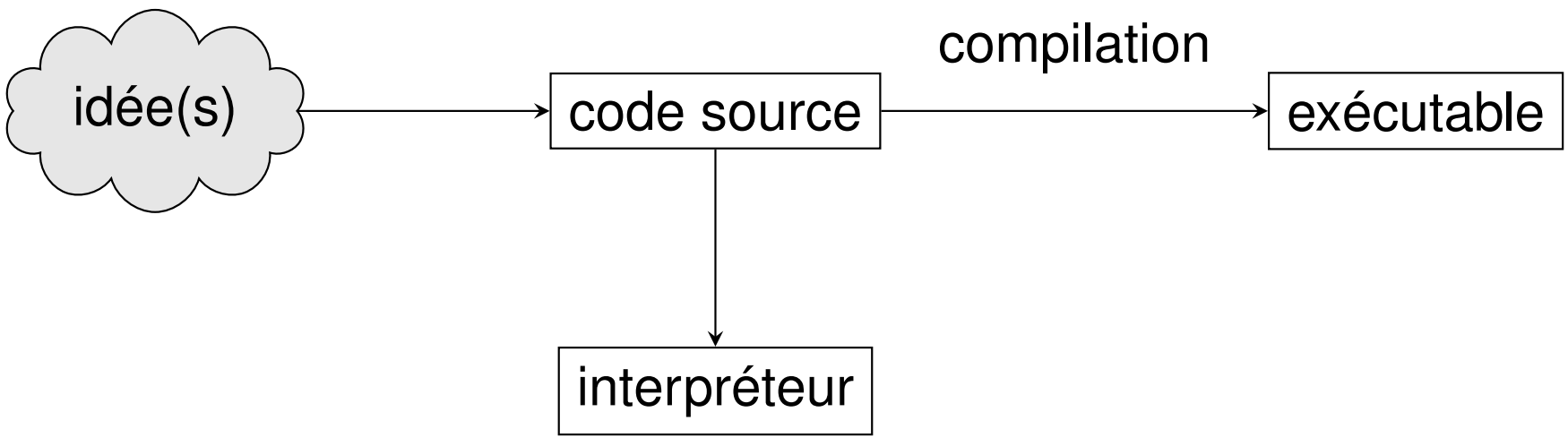
Variables, types et opérations

Instructions et blocs

Instruction conditionnelle (if)

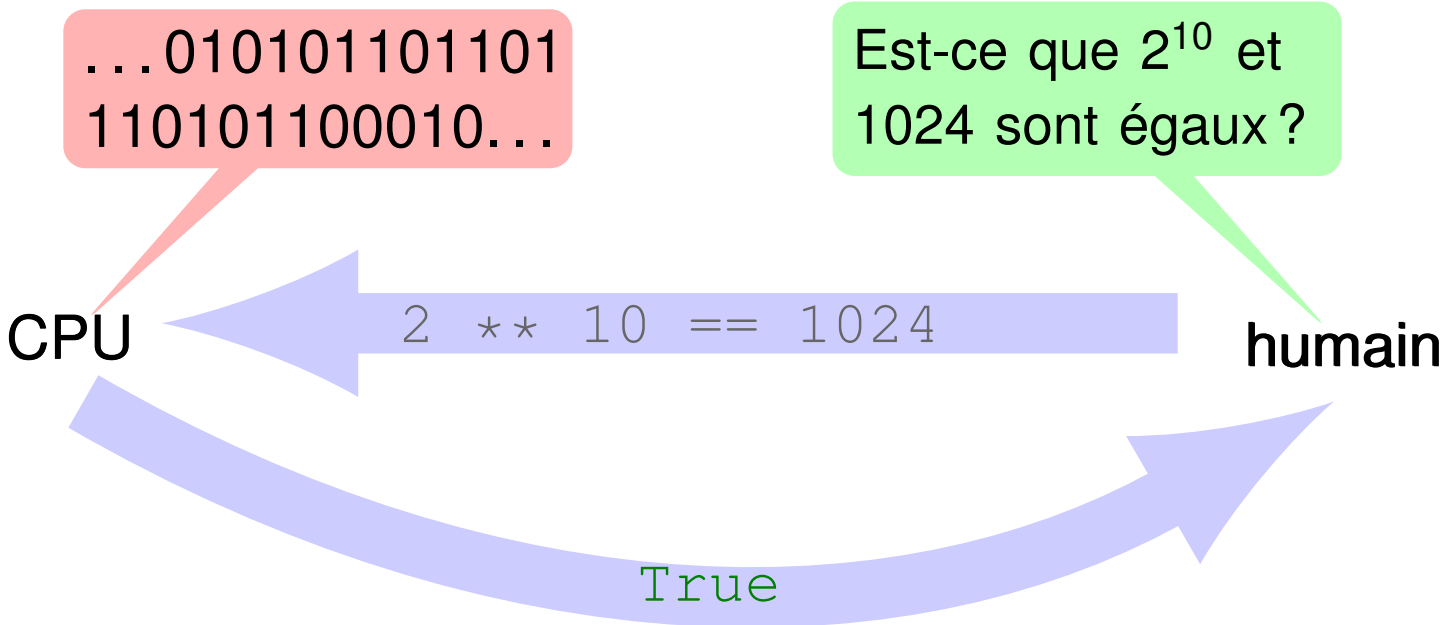
Qu'est-ce que la programmation ?

Programmer : (verbe transitif) "Écrire les programmes informatiques correspondant à l'algorithme de résolution d'un problème" (Larousse), fractionner un problème en instructions codifiées acceptables par la machine.



Pourquoi un langage ?

- L'ordinateur ne comprend que le langage binaire ;
  - trop difficile pour les humains (en général) ;
  - pas compatible d'une machine à l'autre !



- Il nous faut donc un langage commun pour se comprendre (Python, C, Java, ...) ;

Introduction sur un exemple

Un exemple de programme en Python 3 :

```
if __name__ == '__main__':
    date = input(
        "Bonjour! Quelle est votre date "
        "de naissance (jj/mm/aaaa)? "
    )
    jour_naissance = int(date.split("/")[0])
    mois_naissance = int(date.split("/")[1])
    annee_naissance = int(date.split("/")[2])
    age = 2020 - annee_naissance
    if mois_naissance > 9:
        age = age - 1
    if mois_naissance == 9 and jour_naissance > 5:
        age = age - 1
    print("Vous avez", age, "ans")
```

C'est un programme un peu avancé. . . mais essayons de comprendre ce qu'il fait.

La coloration syntaxique

```
if __name__ == '__main__':
    date = input(
        "Bonjour! Quelle est votre date "
        "de naissance (jj/mm/aaaa)? "
    )
    jour_naissance = int(date.split("/")[0])
    mois_naissance = int(date.split("/")[1])
    annee_naissance = int(date.split("/")[2])
    age = 2020 - annee_naissance
    if mois_naissance > 9:
        age = age - 1
    if mois_naissance == 9 and jour_naissance > 5:
        age = age - 1
    print("Vous avez", age, "ans")
```

- Certains mots sont en couleurs : c'est la **coloration syntaxique** ;
- Elle varie selon l'éditeur utilisé ;
- Elle rend le programme plus lisible.

La coloration syntaxique

- La couleur d'un mot indique sa catégorie ; ici, on a entre autres :
  - des fonctions intégrées de Python en vert :

    `input` = entrée , `int` = entier
  - des mots-clés de Python en **vert gras** :

    `if` = si , **`print`** = imprimer (afficher)
  - des opérateurs en **mauve gras** :

    `and` = et
  - des chaînes de caractères en rouge :

    `"Quelle est votre date de naissance?"`

    ⋮
- Nos variables ne sont pas coloriées ;

L'indentation

```
if __name__ == '__main__':
    date = input(
        "Bonjour! Quelle est votre date "
        "de naissance (jj/mm/aaaa)? "
    )
    jour_naissance = int(date.split("/")[0])
    mois_naissance = int(date.split("/")[1])
    annee_naissance = int(date.split("/")[2])
    age = 2020 - annee_naissance
    if mois_naissance > 9:
        age = age - 1
    if mois_naissance == 9 and jour_naissance > 5:
        age = age - 1
    print("Vous avez", age, "ans")
```

- Les lignes ne commencent pas toutes au même endroit.
- Ce n'est pas un détail !
- Le positionnement (on dit l'**indentation**) des lignes est important pour leur signification.

Modifier l'indentation modifie le comportement du programme — et peut même l'empêcher de fonctionner.

La structure d'un programme Python

Les programmes Python doivent être **structurés** (cf. `modele.py` sur la page du cours) :

Modèle

```
def main():
    # mon code ici

if __name__ == "__main__":
    main()
```

On expliquera pourquoi plus tard.

Configurez votre éditeur pour qu'il insère automatiquement ce texte dans vos nouveaux fichiers.

En résumé

- On a vu un exemple avancé illustrant dans les grandes lignes à quoi ressemble un programme ;
- On a vu
  - que les lignes sont des instructions à effectuer, dans l'ordre ;
  - qu'on peut stocker des valeurs (dans `date`, `age`, ...) ;
  - qu'on peut effectuer des calculs (`age - 1`) ;
  - qu'on peut utiliser des fonctions de Python (**`print`**, `input`, `int`) ;
  - que la structure et l'indentation importent ;
  - qu'on peut exécuter le programme `fichier.py` avec la commande `python3 fichier.py`
  - ...

- Dans la suite, on va apprendre pas à pas et dans le détail toutes ces notions (et bien d'autres), afin que vous maîtrisiez les bases de la programmation

Python pas à pas

Variables, types et opérations

Types de valeurs

- Les **valeurs de base** possèdent un **type** ;
- Le type va notamment déterminer ce qui se passe quand on fait une opération sur des valeurs ;

Les principaux types (on en verra d'autres) sont :

Type	**En Python**	**Exemples**
entier	`int`	12, -4, 123545, ...
flottant	`float`	3.14159, -1.5, 12., 4.56e12, ...
booléen	`bool`	`True` (vrai) ou `False` (faux)
indéfini, rien	`None`	`None`
chaîne de caractères	`str`	`'bonjour'`, `'IUT info'`, ...

Les majuscules/minuscules sont importantes :

`True` $\neq$ `true`

Opérations sur les nombres

- Les opérations mathématiques que vous connaissez existent en Python ;
- Le type du résultat dépend du type des données de départ :

$+, -, *$	`int`	`float`
`int`	`int`	`float`
`float`	`float`	`float`

- Attention : la division ($/$) donne toujours un `float` ;
- On a aussi :

math	python	exemple
$x \mod y$	`x % y`	`19 % 4` $\rightarrow 3$
x^y	`x ** y`	`3 ** 3` $\rightarrow 27$
$\lfloor x/y \rfloor$	`x // y`	`19 // 4` $\rightarrow 4$

Opérations sur les booléens

- On a les opérations logiques sur les booléens :

p	q	p **and** q	p **or** q	**not** p
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

- Les comparaisons produisent des booléens :

math	python	math	python
$a = b$?	`a == b`	$a \neq b$?	`a != b`
$a \leq b$?	`a <= b`	$a < b$?	`a < b`
$a \geq b$?	`a >= b`	$a > b$?	`a > b`

Opérations sur les chaînes de caractères

- Seuls **+** et * marchent sur les chaînes ;
  - **+** réalise la **concaténation** de deux chaînes :
    `'IUT' + 'CHAMPS'` → `'IUTCHAMPS'`
  - * concatène `n` exemplaires de la chaîne :
    `'IUT' * 3` → `'IUTIUTIUT'`
- Les comparaisons (<, <=, ...) fonctionnent et se basent sur l'ordre alphabétique (`'a...' < 'b...' < 'c...' < ...`) :
  `'IUT' < 'CHAMPS'` → `False`

- Il existe beaucoup d'autres opérations sur les chaînes ;
- Et aussi plein de fonctions mathématiques (cos, log, . . .) ;

Variables

- Une **variable** est un nom qui référence une valeur en mémoire ;
- On peut s'en servir dans les calculs ;
- Elle a le même type que la valeur qu'elle référence ;

Affectation

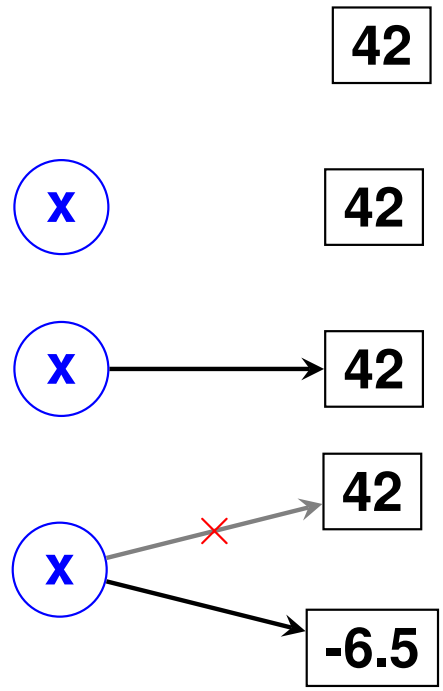
- L'**affectation** d'une variable lie un nom à une valeur ;
- La syntaxe : **nom = expression**, où **expression** est une valeur ou un calcul qui produit une valeur ;

Étapes de l'affectation

```
x = 40 + 2
```

1. On commence par calculer le membre droit, ici on trouve **42**
2. Ensuite on crée le nom pour **x** (sauf s'il a déjà été créé)
3. Enfin on relie la variable à sa valeur

   En cas de réaffectation, le lien d'avant est perdu : `x = -6.5`



Affectation $\neq$ égalité !

Ce n'est pas du tout le **=** des mathématiques. Il faut le lire comme "prend la valeur" : **x = x + 1**

Exemple

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

La situation sera différente pour les "objets modifiables" qu'on verra plus tard.

Nommage

Certaines **règles** doivent être respectées pour nommer les variables (et les fonctions qu'on verra plus tard) :

- caractères autorisés : abc..., ABC..., 0123..., _ ;
- caractères interdits : tout le reste (et é, è, à, ç, ...) ;

- **attention** : les chiffres sont permis, mais pas au début !
  `exemple` `_ex2` `Ex2mpl1` ~~`2020IUT`~~

- **noms** interdits : ce qui a déjà été utilisé, ainsi que les mots réservés et les fonctions prédéfinies de Python (voir suite) ;

Mots réservés

Les mots suivants sont réservés pour le langage :

`and`	`as`	`assert`	`break`	`class`	`continue`
`def`	`del`	`elif`	`else`	`except`	`finally`
`for`	`from`	`global`	`if`	`import`	`in`
`is`	`lambda`	`nonlocal`	`not`	`or`	`pass`
`raise`	`return`	`try`	`while`	`with`	`yield`
`True`	`False`	`None`			

Si votre éditeur affiche le nom de votre variable en gras ou en couleur, c'est qu'il s'agit d'un mot réservé $\Rightarrow$ choisissez un autre nom !

Au passage ... `input`

```
date = input(
    "Bonjour! Quelle est votre date "
    "de naissance (jj/mm/aaaa)? "
)
```

- `input(str)` permet à l'utilisateur de rentrer des valeurs ;
- Ici, on lui demande de rentrer une date ;
- Attention : la fonction `input` renvoie **toujours** du texte !

Exemple

```
>>> x = input("Entrez un nombre: ")
Entrez un nombre: 1
>>> y = input("Entrez un deuxième nombre: ")
Entrez un deuxième nombre: 3
>>> print(x, "+", y, "=", x + y)
1 + 3 = 13
```

Transtypage (conversion de types)

- Pour convertir ce que nous donne `input` en entier, on utilise la fonction `int` ;
- On peut donc corriger l'exemple précédent comme ceci :

Exemple

```
>>> x = input("Entrez un nombre: ")
Entrez un nombre: 1
>>> y = input("Entrez un deuxième nombre: ")
Entrez un deuxième nombre: 3
>>> x = int(x)   # conversion de la chaîne x en entier
>>> y = int(y)   # conversion de la chaîne y en entier
>>> print(x, "+", y, "=", x + y)
1 + 3 = 4
```

- Plus généralement :
  - `str(valeur)` convertit `valeur` en chaîne ;
  - `int(valeur)` convertit `valeur` en entier **si possible** ;
  - `float(valeur)` convertit `valeur` en réel **si possible** ;
  - `bool(valeur)` convertit `valeur` en booléen ;

Quelques exemples

Voici quelques exemples de conversions :

```
int(4.5) → 4          int(-4.5) → -4        int('0345') → 345
int('IUT') → erreur   float(4) → 4.         float('4.5') → 4.5
str(4) → '4'          str(True) → 'True'    str(-4.5) → '-4.5'
bool(4) → True        bool(0) → False       bool('IUT') → True
bool(None) → False    int('4.5') → erreur
```

Remarques importantes :

- `str` fonctionne toujours ;
- `int` et `float` échouent sur une chaîne qui n'est pas un nombre ;
- `int` échoue sur une chaîne représentant un réel ;
  - solution : `int(float('4.5'))` au lieu de `int('4.5')` ;
- `bool` donne `False` si son paramètre équivaut à 0, `True` sinon

La fonction `type`

- On connaît en général le type des variables qu'on définit soi-même ;
- Comment faire pour connaître le type des variables qui viennent d'ailleurs ?
- On utilise la fonction `type` ;

Exemple

```
>>> type(1.5)
<class 'float'>
>>> type(2)
<class 'int'>
>>> type(input)
<class 'builtin_function_or_method'>
```

`TypeError` : types incompatibles

- Certains types sont incompatibles : on ne peut pas les mélanger dans une même opération ;
- Quand ce problème survient, Python provoque une **exception** de type `TypeError` ;
- Cela a pour effet d'arrêter le programme ;

Exemple

```
>>> '3' * 2    # ok
'33'
>>> '3' + 2    # erreur!
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

- Si une variable n'a pas le bon type, vous devez la convertir (transtypage) ;

Python pas à pas

Instructions et blocs

Instructions et séquences d'instructions

```
if __name__ == '__main__':
    date = input(
        "Bonjour! Quelle est votre date "
        "de naissance (jj/mm/aaaa)? "
    )
    jour_naissance = int(date.split("/")[0])
    mois_naissance = int(date.split("/")[1])
    annee_naissance = int(date.split("/")[2])
    age = 2020 - annee_naissance
    if mois_naissance > 9:
        age = age - 1
    if mois_naissance == 9 and jour_naissance > 5:
        age = age - 1
    print("Vous avez", age, "ans")
```

- Les instructions sont exécutées dans l'ordre, de haut en bas
- En Python, il n'y a **qu'une instruction par ligne**
- Le flot d'instructions peut être modifié / redirigé par des conditions (`if`), des boucles (plus tard), ...

Blocs d'instructions

Certaines instructions sont regroupées en **blocs** de la façon suivante :

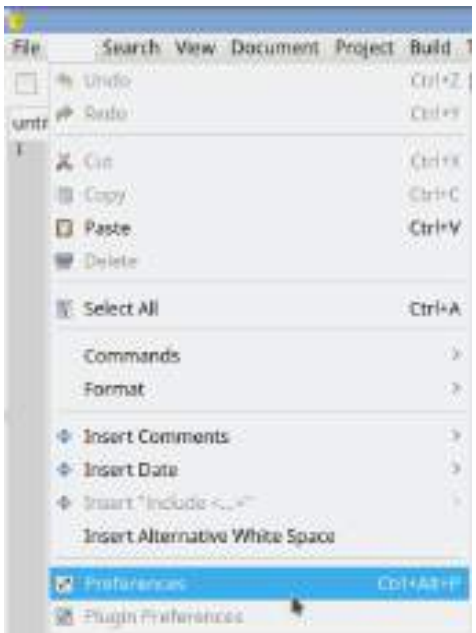
```
entête du bloc:
    instruction 1 du bloc
    instruction 2 du bloc
    instruction 3 du bloc
instruction hors bloc
```

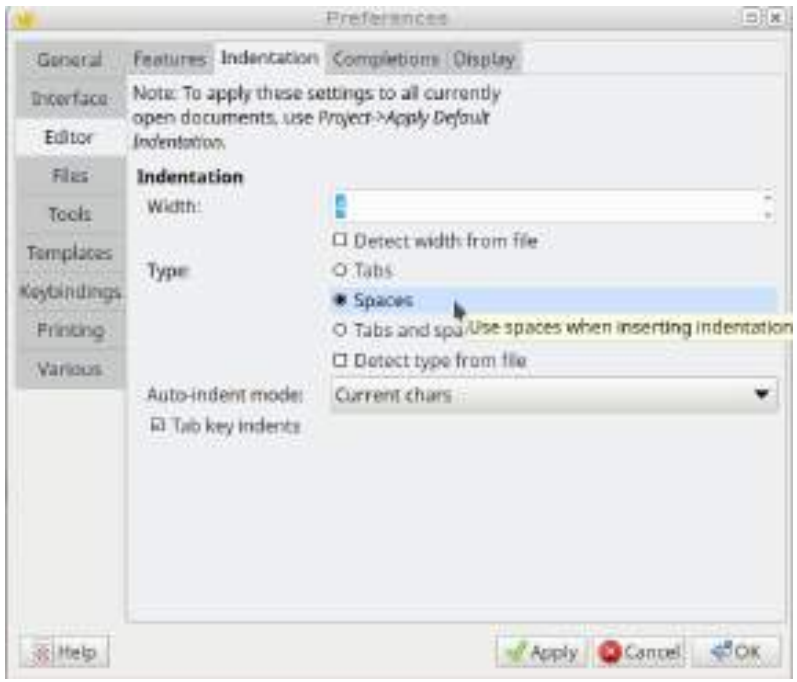
- L'indentation (le décalage) se fait avec la touche tabulation ;
- On peut insérer un bloc dans un bloc, un bloc dans un bloc dans un bloc, ...
- L'indentation fait partie du langage Python, changer l'indentation **change la signification** du programme

Espaces, pas tabulations !

Ne mélangez pas tabulations et espaces dans l'indentation !!! Configurez votre éditeur pour qu'il remplace les tabulations par des espaces.

Sous Geany :





Python pas à pas

Instruction conditionnelle (if)

La conditionnelle : le `if`

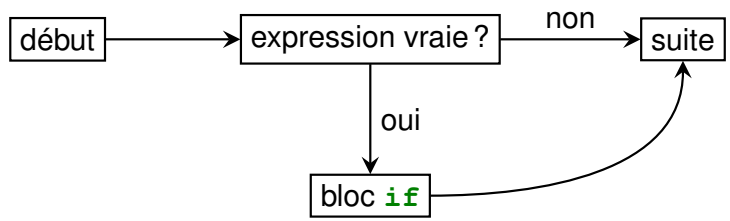
```
if mois_naissance > 9:
    age = age - 1
if mois_naissance == 9 and jour_naissance > 5:
    age = age - 1
print("Vous avez", age, "ans")
```

- Dans cet exemple, on teste si `mois_naissance > 9` ;
  - Si c'est le cas, on effectue le bloc qui suit, où `age` chute d'une unité
- On teste ensuite si `mois_naissance == 9` ET si `jour_naissance > 5` ;
  - Si c'est le cas, on retire une unité à `age`
- Dans tous les cas, on termine par afficher l'âge calculé ;

La conditionnelle : le `if`

La forme la plus simple est

```
# début
if expression:
    # instruction 1 du if
    # instruction 2 du if
    # ...
# suite
```



- **expression** est une expression qui renvoie un booléen, qui est donc évaluée à `True` ou `False`
- les instructions du bloc du `if` sont effectuées **uniquement si** l'expression est évaluée à `True`
- dans tous les cas, le programme reprend à **l'instruction après `if`**

Conditions multiples

- L'expression du `if` peut contenir plusieurs conditions ;

```
if jour == "mardi" and heure > 16 or minutes < 30:
```

- Les conditions sont évaluées dans l'ordre de lecture ;
- Python ne teste pas tout si le résultat est connu d'avance :

`1 < 2 or x / 0 == 3` ⟶ `True` (même si `x` n'existe pas)

`x / 0 == 3 or 1 < 2` ⟶ **erreur**

Attention : Python 2 $\neq$ Python 3

Les deux versions **ne sont pas** compatibles !

Quelques liens utiles

Indispensable

- Télécharger Python : `http://www.python.org/`

Editeurs

Les éditeurs suivants rendent le développement Python plus facile ; choisissez celui que vous préférez :

Geany	`https://www.geany.org/`
PyCharm	`https://www.jetbrains.com/pycharm/`
Notepad++	`https://notepad-plus-plus.org/`
Kate	`http://kate-editor.org/`
SublimeText	`https://www.sublimetext.com/`

Autres

- Visualiser Python : `http://pythontutor.com/`