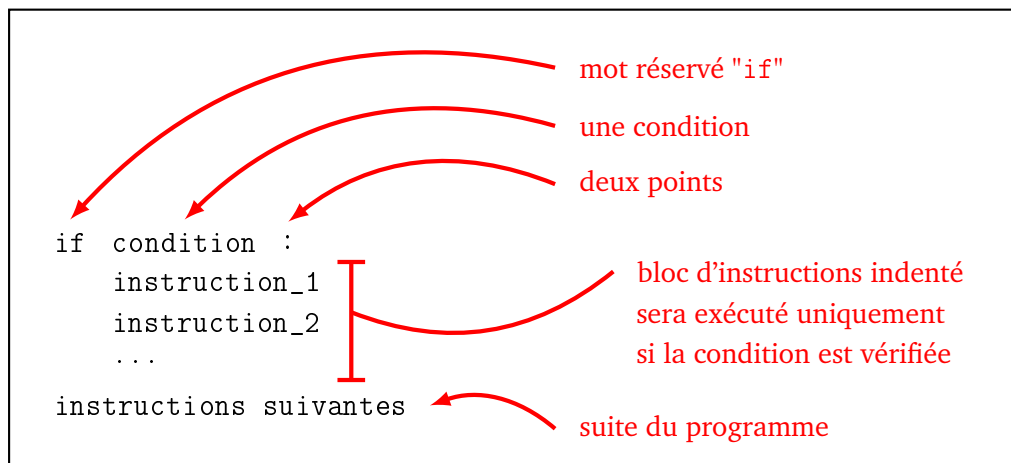
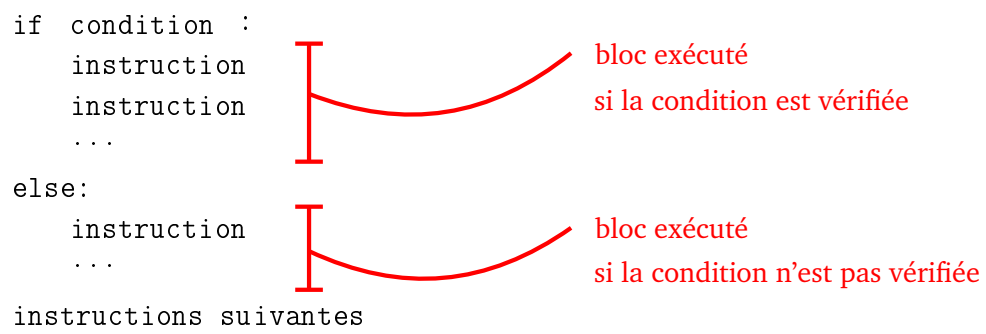


Si ... alors ...



Si ... alors ... sinon ...

```
if condition :  
    instruction  
    instruction  
    ...  
else:  
    instruction  
    ...  
instructions suivantes
```



The diagram illustrates the execution flow of an if-else statement. Two red vertical bars with horizontal caps are positioned to the right of the code blocks. The top bar spans the 'if' block, and the bottom bar spans the 'else' block. Red curved arrows originate from the right side of these bars and point to the right. The top arrow points to the text 'bloc exécuté si la condition est vérifiée'. The bottom arrow points to the text 'bloc exécuté si la condition n'est pas vérifiée'.

bloc exécuté
si la condition est vérifiée

bloc exécuté
si la condition n'est pas vérifiée

Entrée au clavier

- `input()` met en pause l'exécution du programme et attend de l'utilisateur un texte
- Cette commande renvoie une chaîne de caractères.
- `int`. Si on veut un entier, il faut convertir la chaîne.
Exemple. Si `age_chaine` vaut "17", alors `int(age_chaine)` vaut l'entier 17.
- `float`. Si on veut un nombre flottant, il faut convertir la chaîne.
Exemple. Si `pi_chaine` vaut "3.14", alors `float(pi_chaine)` vaut le nombre flottant 3.14.
- `str` convertit un nombre en une chaîne.
Exemple. `str(17)` renvoie la chaîne "17"; si `age = 17`, alors `str(age)` renvoie également "17".

Le module « random »

Le module `random` génère des nombres comme s'ils étaient tirés au hasard.

- En début du programme :

```
from random import *
```

- `randint(a,b)` renvoie un entier au hasard compris entre a et b .
Exemple. `n = randint(1,6)`, n est un entier tiré au hasard avec $1 \leq n \leq 6$. Si on recommence l'instruction `n = randint(1,6)`, n prend une nouvelle valeur. C'est comme si on effectuait le lancer d'un dé à 6 faces.
- `random()` renvoie un nombre flottant compris entre 0 et 1.
Exemple. Avec `x = random()`, alors x est un nombre flottant avec $0 \leq x < 1$.

Booléens

- Un **booléen** est une donnée qui vaut soit la valeur « vrai », soit la valeur « faux ». En Python les valeurs sont `True` et `False` (avec une majuscule).
- On obtient un booléen par exemple comme résultat de la comparaison de deux nombres. Par exemple `7 < 4` vaut `False` (car 7 n'est pas plus petit que 4). Vérifie que `print(7 < 4)` affiche `False`.

Voici les principales comparaisons :

- **Test d'égalité** : `a == b`
- **Test inférieur strict** : `a < b`
- **Test inférieur large** : `a <= b`
- **Test supérieur** : `a > b` ou `a >= b`
- **Test non égalité** : `a != b`

Par exemple `6*7 == 42` vaut `True`.

Booléens

ATTENTION ! L'erreur classique est de confondre « `a = b` » et « `a == b` ».

- **Affectation.** `a = b` met le contenu de la variable `b` dans la variable `a`.
- **Test d'égalité.** `a == b` teste si les contenus de `a` et de `b` sont égaux et vaut `True` ou `False`.

Booléens

- On peut comparer autre chose que des nombres. Par exemple « `car == "A"` » teste si la variable `car` vaut "A"; « `il_pleut == True` » teste si la variable `il_pleut` est vraie...
- Les booléens sont utiles dans le test « si ... alors ... » et dans les boucles « tant que ... alors ... ».
- **Opérations entre les booléens.** Si P et Q sont deux booléens, on peut définir de nouveaux booléens.
 - **Et logique.** « $P \text{ and } Q$ » est vrai si et seulement si P et Q sont vrais.
 - **Ou logique.** « $P \text{ or } Q$ » est vrai si et seulement si P ou Q est vrai.
 - **Négation.** « $\text{not } P$ » est vrai si et seulement si P est faux.

Exemple. « $(2+2 == 2*2) \text{ and } (5 < 3)$ » renvoie `False`, car même si on a bien $2+2 = 2 \times 2$, l'autre condition n'est pas remplie car $5 < 3$ est faux.