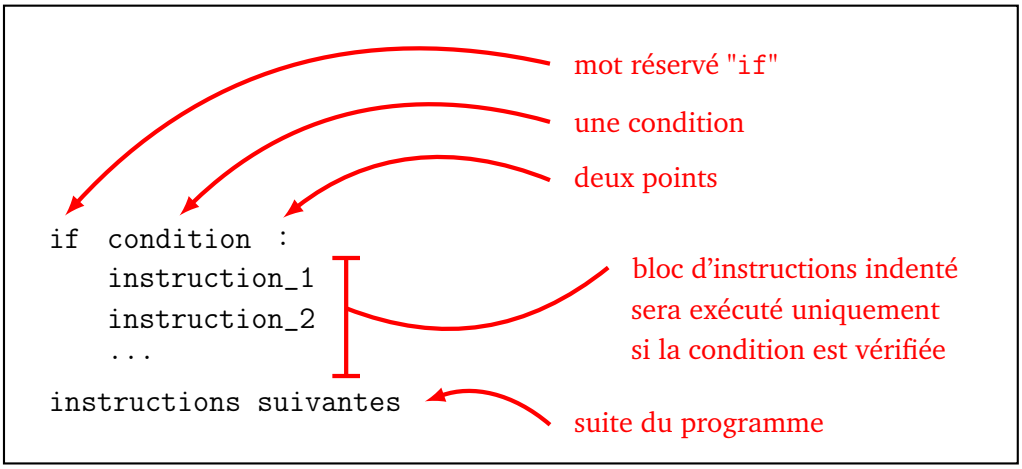
Si ... alors ...

L'ordinateur peut réagir en fonction d'une situation. Si une condition est remplie il agit d'une certaine façon, sinon il fait autre chose.

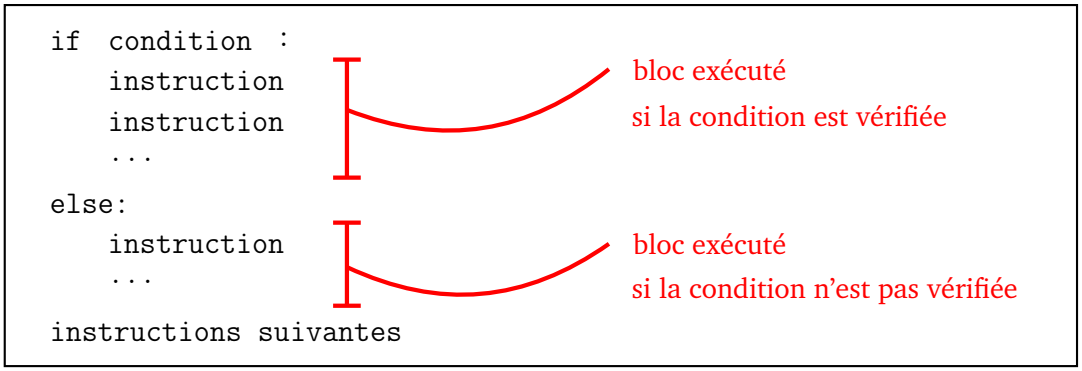
Cours 1 (Si ... alors ...)**.**
Voici comment utiliser le test « `if` » avec `Python` :



Voici un exemple, qui avertit un conducteur si une variable `vitesse` est trop grande.

```
if vitesse > 110:
    print("Attention, tu roules trop vite.")
```

On peut aussi exécuter des instructions si la condition n'est pas remplie à l'aide du mot « `else` ».



Encore une fois c'est l'indentation qui délimite les différents blocs d'instructions. Voici un exemple qui affiche le signe d'un nombre `x`.

```
if x >= 0:
    print("Le nombre est positif.")
else:
    print("Le nombre est négatif.")
```

Cours 2 (Entrée au clavier)**.**
Pour pouvoir interagir avec l'utilisateur, tu peux lui demander de saisir un texte au clavier. Voici un petit programme qui demande le prénom et l'âge de l'utilisateur et affiche un message du style « Bonjour Kevin » puis « Tu es mineur/majeur ! » selon l'âge.

```
prenom = input("Comment t'appelles-tu ? ")
print("Bonjour",prenom)

age_chaine = input("Quel âge as-tu ? ")
age = int(age_chaine)

if age >= 18:
    print("Tu es majeur !")
else:
    print("Tu es mineur !")
```

Explications.

- La commande `input()` met en pause l'exécution du programme et attend de l'utilisateur un texte (qu'il termine en appuyant sur la touche « Entrée »).
- Cette commande renvoie une chaîne de caractères.
- Si on veut un entier, il faut convertir la chaîne. Par exemple, ici `age_chaine` peut valoir `"17"` (ce n'est pas un nombre mais une suite de caractères), alors que `int(age_chaine)` vaut maintenant l'entier 17.
- L'opération inverse est aussi possible, `str()` convertit un nombre en une chaîne. Par exemple `str(17)` renvoie la chaîne `"17"` ; si `age = 17`, alors `str(age)` renvoie également `"17"`.

Cours 3 (Le module « random »)**.**
Le module `random` génère des nombres comme s'ils étaient tirés au hasard.

- Voici la commande à placer au début du programme pour appeler ce module :
  ```
  from random import *
  ```
- La commande `randint(a,b)` renvoie un entier au hasard compris entre a et b.
  Par exemple après l'instruction `n = randint(1,6)`, n est un entier tiré au hasard avec $1 \leqslant n \leqslant 6$.
  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.
- La commande `random()`, sans argument, renvoie un nombre flottant (c'est-à-dire un nombre à virgule) compris entre 0 et 1. Par exemple, après l'instruction `x = random()`, alors x est un nombre flottant avec $0 \leqslant x < 1$.

Activité 1 (Quiz multiplications)**.**

Objectifs : programmer un petit test sur les tables de multiplication.

- Définis une variable a, à laquelle tu affectes une valeur au hasard entre 1 et 12.
- Même chose pour une variable b.
- Affiche à l'écran la question : « Combien vaut le produit $a \times b$? » (Remplace a et b par leur valeur !)
- Récupère la réponse de l'utilisateur et transforme-la en un entier.
- Si la réponse est correcte affiche « Bravo ! », sinon affiche « Perdu ! La bonne réponse était... ».

Test d'égalité. Pour tester si deux nombres x et y sont égaux, l'instruction est :

`if x == y:`

Le test d'égalité s'écrit bien avec le double signe égal « == ». Par exemple « `x == 3` » renvoie « vrai » si x vaut 3 et « faux » sinon.

Attention ! La commande « `x = 3` » n'a rien à voir, cette instruction stocke 3 dans la variable x.

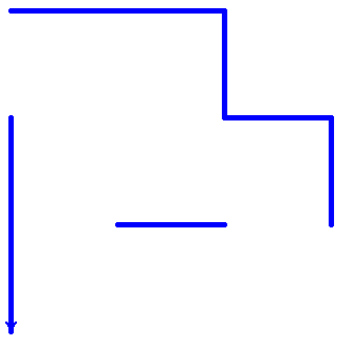
Activité 2 (Les mots de la tortue)**.**

Objectifs : piloter la tortue par un mot, chaque caractère correspondant à une instruction.

Tu donnes un mot à la tortue `Python`, par exemple **AgAdaagAdaA**, dans lequel chaque caractère (lus de gauche à droite) correspond à une instruction que la tortue doit exécuter.

- **A** : avance de 100 en traçant,
- **a** : avance de 100 sans tracer,
- **g** : tourne à gauche de 90 degrés,
- **d** : tourne à droite de 90 degrés.

Exemple. Voici le dessin que doit tracer la tortue lorsque `mot` vaut `"AagAgAdAgAAgaAA"`.



Indications. Voici comment parcourir les lettres d'un mot et tester si une lettre est le caractère **A** :

```
for c in mot:
    if c == "A":
        instructions...
```

Cours 4 (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`.
- **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`.
- 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 and Q` » est vrai si et seulement si P et Q sont vrais.
  — **Ou logique.** « `P or Q` » est vrai si et seulement si P ou Q est vrai.
  — **Négation.** « `not P` » est vrai si et seulement si P est faux.
  Par exemple « `(2+2 == 2*2) 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.

Activité 3 (Chiffres d'un nombre).

Objectifs : trouver des nombres dont les chiffres vérifient certaines propriétés.

1. Le programme suivant affiche tous les entiers de 0 à 99. Comprends ce programme. Que représentent les variables u et d ?

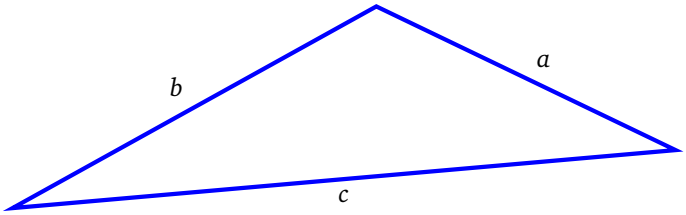
```
for d in range(10):
    for u in range(10):
        n  = 10*d + u
        print(n)
```

2. Trouve tous les entiers compris entre 0 et 999 qui vérifient toutes les propriétés suivantes :
   - l'entier se termine par 3,
   - la somme des chiffres est supérieure ou égale à 15,
   - le chiffre des dizaines est pair.
3. Modifie ton programme précédent pour compter et afficher le nombre d'entiers vérifiant les propriétés.

Activité 4 (Triangles)**.**

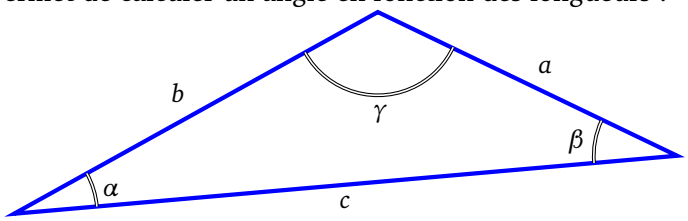
Objectifs : déterminer les propriétés d'un triangle à partir des trois longueurs des côtés.

On se donne trois longueurs a, b et c. Tu vas déterminer les propriétés du triangle dont les longueurs seraient a, b, c.



Définis trois variables a, b et c avec des valeurs entières et $a \leqslant b \leqslant c$ (ou bien demande à l'utilisateur trois valeurs).

1. **Ordre.** Demande à `Python` de tester si les longueurs vérifient bien $a \leqslant b \leqslant c$. Affiche une phrase pour la réponse.
2. **Existence.** Il existe un triangle correspondant à ces longueurs si et seulement si :
$$a + b \geqslant c.$$
Demande à `Python` de tester si c'est le cas et affiche la réponse.
3. **Triangle rectangle.** Demande à `Python` de tester si le triangle est un triangle rectangle. (Pense au théorème de Pythagore.)
4. **Triangle équilatéral.** Teste si le triangle est équilatéral.
5. **Triangle isocèle.** Teste si le triangle est isocèle.
6. **Angles aigus.** Teste si tous les angles sont aigus (c'est-à-dire inférieurs ou égaux à 90 degrés).
Indications.
   - La loi des cosinus permet de calculer un angle en fonction des longueurs :



$$\cos\alpha = \frac{-a^2+b^2+c^2}{2bc}, \qquad \cos\beta = \frac{a^2-b^2+c^2}{2ac}, \qquad \cos\gamma = \frac{a^2+b^2-c^2}{2ab}.$$

   - Pour tester si l'angle α est aigu il suffit de vérifier $\cos\alpha \geqslant 0$ (au final on ne calcule jamais α, mais juste $\cos\alpha$).

Trouve des exemples de longueurs a, b, c pour illustrer les différentes propriétés.

Activité 5 (Le nombre mystère)**.**

Objectifs : coder le jeu incontournable lorsque l'on apprend à programmer. L'ordinateur choisit un nombre au hasard. L'utilisateur doit deviner ce nombre en suivant des indications « plus grand » ou « plus petit » données par l'ordinateur. Comme ce jeu est vite lassant, on introduit des variantes où l'ordinateur a le droit de mentir ou de tricher !

1. **Le jeu classique.**
   - L'ordinateur choisit au hasard un nombre mystère entre 0 et 99.
   - Le joueur propose une réponse.
   - L'ordinateur répond « le nombre à trouver est plus grand » ou « le nombre à trouver est plus petit » ou « bravo, c'est le bon nombre ! ».
   - Le joueur a sept tentatives pour trouver la bonne réponse.

   Programme ce jeu !

   Indications. Pour quitter une boucle `for` avant la dernière proposition, tu peux utiliser la commande `break`. Utilise ceci lorsque le joueur trouve la bonne réponse.

2. **L'ordinateur ment.**

   Pour compliquer le jeu, l'ordinateur a le droit de mentir de temps en temps. Par exemple environ une fois sur quatre l'ordinateur donne la mauvaise indication « plus grand » ou « plus petit ».

   Indications. Pour décider quand l'ordinateur ment, à chaque tour tire un nombre au hasard entre 1 et 4, si c'est 4 l'ordinateur ment !

3. **L'ordinateur triche.**

   Maintenant l'ordinateur triche (mais il ne ment plus) ! À chaque tour l'ordinateur change un peu le nombre mystère à trouver.

   Indications. À chaque tour, tire un nombre au hasard, entre -3 et $+3$ par exemple, et ajoute-le au nombre mystère. (Attention à ne pas dépasser les bornes 0 et 99.)