Conditions en Python

Activité – Première NSI

① Pourquoi des conditions ?

Exercice 1 *Trois cas, trois réactions*

Un programme doit indiquer si une note sur 20 correspond à « admis » ($\geqslant 10$), « à repasser » (entre 8 inclus et 10 exclu) ou « refusé » (strictement inférieur à 8).

1. Pour les notes 15, 9 et 6, indiquer dans chaque cas le résultat attendu par le programme.
2. Sans avoir encore vu `if` en Python, comment décririez-vous l'algorithme à un humain qui ne l'a jamais vu, en utilisant les expressions « si », « sinon si », « sinon » ?

Remarque 1. Une **condition** permet d'exécuter un bloc d'instructions *seulement si* une expression booléenne est vraie. C'est la base de tous les programmes qui prennent une décision.

② Expressions booléennes

Définition 1 **(Booléen)** Un **booléen** est une valeur qui ne peut prendre que deux états : `True` (vrai) ou `False` (faux). Les comparaisons (`<`, `<=`, `==`, `!=`, `>`, `>=`) produisent des booléens.

Exercice 2 *Évaluer des expressions booléennes*

Sans exécuter Python, donner la valeur de chaque expression. Justifier brièvement.

1. `5 > 3`
2. `5 == 5.0`
3. `"abc" == "ABC"`
4. `10 / 2 == 5`
5. `3 < 5 < 10`
6. `not (3 == 4)`
7. `(2 + 2 == 4) and (2 + 2 == 5)`
8. `(2 + 2 == 4) or (2 + 2 == 5)`

Propriété 1 **(Tables de vérité)**

A	B	A and B	A or B	not A
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

③ La structure `if` / `else`

Définition 2 (`if` simple, `if/else`, `if/elif/else`)

```
if condition_1:
    bloc_1
elif condition_2:
    bloc_2
else:
    bloc_par_defaut
```

Python évalue les conditions *dans l'ordre* : dès que l'une est vraie, le bloc correspondant est exécuté et les autres sont ignorés. Le `else` (optionnel) capture tous les cas restants.

Exercice 3 *Le programme du début*

Écrire la fonction `mention(note)` qui prend une note sur 20 en paramètre et renvoie une chaîne de caractères :

— `"admis"` si la note est supérieure ou égale à 10 ;

— `"à repasser"` si la note est dans [8 ; 10[;

— `"refusé"` sinon.

Tester avec les notes 15, 9 et 6.

Exercice 4 *Repérer une erreur*

Voici une fonction « buggée » qui devrait renvoyer la valeur absolue d'un nombre :

```
1 def valeur_absolue(x):
2     if x > 0:
3         return x
4     if x < 0:
5         return -x
```

1. Pour quelle valeur de x cette fonction est-elle incorrecte ? Pourquoi ?
2. Proposer deux corrections différentes.

4 Conditions composées

Exercice 5 *Combiner plusieurs conditions*

Pour chaque problème, écrire une fonction Python qui renvoie un booléen.

1. `est_majeur_et_francais(age, nationalite)` qui renvoie `True` si la personne a au moins 18 ans **et** est de nationalité française.
2. `est_jour_chome(jour)` qui renvoie `True` si le `jour` (chaîne en minuscules) est `"samedi"` **ou** `"dimanche"`.
3. `est_dans_intervalle(x, a, b)` qui renvoie `True` si $a \leqslant x \leqslant b$.
4. `est_bissextile(annee)` qui renvoie `True` si `annee` est bissextile. *Rappel : une année est bissextile si elle est divisible par 4, sauf si elle est divisible par 100, sauf si elle est divisible par 400.*

5 Imbriquer des conditions

Exercice 6 *Tarif d'un parc d'attractions*

Le tarif d'entrée d'un parc d'attractions est calculé ainsi :

— enfant (moins de 12 ans) : gratuit le mercredi, 5 € les autres jours ;

— adolescent (12 à 17 ans) : 10 € le mercredi, 15 € les autres jours ;

— adulte (18 ans ou plus) : 20 € le mercredi, 25 € les autres jours.

Écrire la fonction `tarif(age, jour)` qui renvoie le prix d'entrée. Le paramètre `jour` est une chaîne, par exemple `"mercredi"`.

6 Pièges fréquents

Remarque 2 (= ou == ?).

— `=` est l'**affectation** : `x = 5` affecte la valeur 5 à `x`.
— `==` est la **comparaison** : `x == 5` renvoie `True` si `x` vaut 5, `False` sinon.

Confondre les deux est une erreur très fréquente. En Python, écrire `if x = 5:` déclenche immédiatement une `SyntaxError`, ce qui aide à repérer la faute.

Exercice 7 *Identifier les erreurs*

Pour chaque ligne, indiquer si elle est correcte. Si elle est fausse, expliquer pourquoi et proposer la correction.

1. `if note = 10:`
2. `if note == 10`
3. `if note >= 10:`
4. `if note >= 10 and note <= 20:`
5. `if 10 <= note <= 20:`
6. `if note >= 10 or 20:`

7 Ce qu'il faut retenir

Propriété 2 **(Structure conditionnelle)** `if condition:` exécute le bloc indenté qui suit *seulement si* la condition est vraie. `elif` et `else` (optionnels) gèrent les cas suivants. Python évalue les conditions dans l'ordre et s'arrête à la première vraie.

Propriété 3 **(Opérateurs booléens)** `and` (vrai si les deux opérandes sont vrais), `or` (vrai si au moins l'un l'est), `not` (inverse). Ces opérateurs respectent les tables de vérité classiques.

Propriété 4 **(Comparaison vs affectation)** `x = 5` *affecte* 5 à `x` ; `x == 5` *compare* `x` et 5. Dans une condition, c'est toujours la comparaison `==` (ou `!=`, `<`, etc.) qu'il faut utiliser.

Remarque 3. Une bonne pratique : si une fonction renvoie un booléen, écrire directement `return condition` plutôt que `if condition: return True else: return False`. La forme courte est plus lisible et moins sujette à erreurs.