Chapitre 1

Programmer en Python

Objectifs du chapitre

Domaine : Notions transversales de programmation.
Variables et types, entrées et sorties, f-strings, instructions conditionnelles, boucles bornées et non bornées, fonctions.
Organisation : programme officiel, cours, méthodes, exemples en Python, exercices corrigés et fiche de synthèse.

Programme officiel (BO)

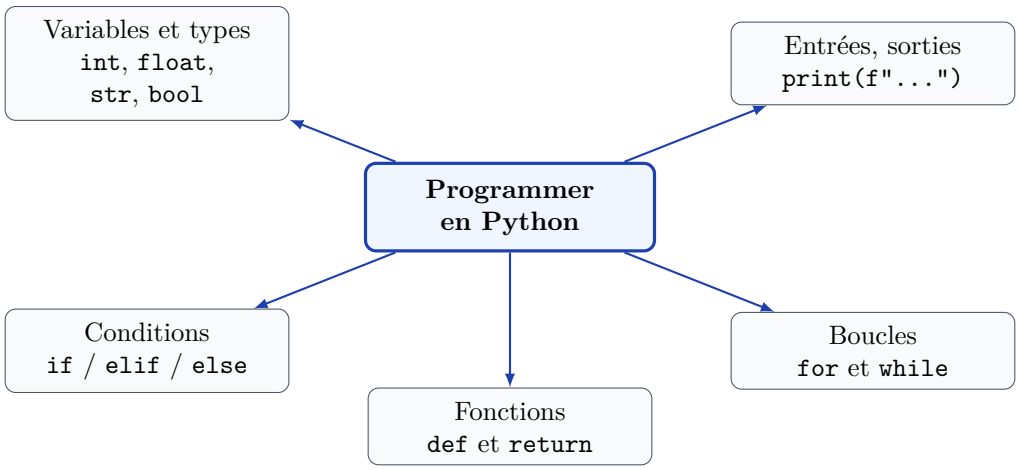
Notions transversales de programmation – Capacités attendues

- Écrire et développer des programmes pour répondre à des problèmes et modéliser des phénomènes physiques, économiques et sociaux.
- Affectations, variables : utiliser des variables de type entier, flottant, chaîne de caractères et booléen.
- Séquences d'instructions : écrire une suite d'instructions exécutées dans l'ordre.
- Instructions conditionnelles : `if`, `elif`, `else`.
- Boucles bornées (`for`) et non bornées (`while`).
- Définitions et appels de fonctions.

Source : BO spécial n°1 du 22 janvier 2019, programme de Sciences numériques et technologie de Seconde générale et technologique.

1.1 Carte mentale du chapitre

Carte mentale



1.2 Introduction

Un **algorithme** est une suite finie d'instructions permettant de résoudre un problème. Un **programme** est la traduction d'un algorithme dans un langage compréhensible par un ordinateur. En SNT, on utilise le langage **Python**, simple à lire et très utilisé (sciences, web, intelligence artificielle, traitement de données...).

Ce chapitre présente les notions de programmation qui serviront dans tous les thèmes de l'année : Internet, le Web, les données structurées, la photographie numérique, la géolocalisation, les réseaux sociaux et les objets connectés.

Attention

En Python, l'**indentation** (le décalage de 4 espaces en début de ligne) n'est pas décorative : elle indique quelles instructions font partie d'un bloc (`if`, `for`, `while`, `def`). Une indentation incorrecte provoque une erreur ou un mauvais résultat.

1.3 Variables, types et affectation

Définition – Variable

Une **variable** est un nom qui désigne une valeur stockée dans la mémoire de l'ordinateur. On lui donne une valeur par une **affectation**, notée `=` en Python : `nom = valeur`.

```
1 age = 15          # on affecte la valeur 15 a la variable age
2 age = age + 1     # on calcule age + 1, puis on range le resultat dans age
3 print(age)        # affiche 16
```

Attention

Le signe `=` n'est pas une égalité mathématique : `age = age + 1` signifie « calculer `age + 1` puis ranger le résultat dans `age` ». Pour **tester** une égalité, on écrit `==`.

Définition – Les quatre types de base

— `int` (entier) : `12`, `-3`, `0` ;
— `float` (flottant, nombre à virgule) : `3.14`, `-0.5` – on utilise un **point**, pas une virgule ;
— `str` (chaîne de caractères) : `"Bonjour"`, `'SNT'` – entre guillemets ;
— `bool` (booléen) : `True` ou `False`.

La fonction `type()` donne le type d'une valeur : `type(3.14)` renvoie `float`.

Propriété – Opérations usuelles

— Sur les nombres : `+`, `-`, `*`, `/` (division décimale), `//` (quotient entier), `%` (reste), `**` (puissance).
— Sur les chaînes : `+` concatène (`"S" + "NT"` donne `"SNT"`), `*` répète (`"ha" * 3` donne `"hahaha"`), `len()` donne la longueur.

Méthode – Suivre l'évolution des variables (tableau d'état)

Pour comprendre un programme, on note la valeur de chaque variable après chaque ligne.

```
1 a = 5
2 b = 3
3 a = a + b
4 b = a - b
```

Après la ligne	1	2	3	4
`a`	5	5	8	8
`b`	–	3	3	5

1.4 Entrées, sorties et f-strings

Définition – `print` et `input`

`print(...)` **affiche** une ou plusieurs valeurs à l'écran. `input("message")` **demande** une saisie à l'utilisateur et renvoie toujours une **chaîne de caractères**.

```
1 prenom = input("Quel est ton prenom ? ")
2 annee = int(input("Ton annee de naissance ? "))   # conversion en entier
3 print(f"Bonjour {prenom} ! Tu auras {2026 - annee} ans cette annee.")
```

Attention

Sans `int(...)` ou `float(...)`, le calcul `2026 - annee` provoque une erreur car on ne peut pas soustraire une chaîne à un nombre.

Définition – Chaîne formatée (f-string)

Une **f-string** est une chaîne de caractères précédée de la lettre `f` : `f"..."`. Tout ce qui est écrit entre accolades `{ }` est **remplacé par sa valeur** : une variable, un calcul ou un appel de fonction.

```
1 nom = "Lina"
2 note = 15.5
```

```
print(f"{nom} a obtenu {note}/20")          # Lina a obtenu 15.5/20
print(f"Le double de 7 vaut {2 * 7}")       # Le double de 7 vaut 14
print(f"{nom} contient {len(nom)} lettres") # Lina contient 4 lettres
```

Propriété – Arrondir à l'affichage

Dans les accolades, on peut préciser un format après deux-points : `{x:.2f}` affiche `x` avec **2 chiffres après la virgule**, sans modifier la valeur de `x`.

```
prix = 19.0 / 3
print(f"Prix unitaire : {prix:.2f} euros")    # Prix unitaire : 6.33 euros
```

Méthode – Remplacer les virgules de `print` par une f-string

Au lieu de `print("Il a", age, "ans")`, on écrit `print(f"Il a {age} ans")` : on rédige la phrase entière, puis on place chaque valeur entre accolades à l'endroit voulu. Les espaces sont exactement ceux que l'on tape.

Attention

Sans le `f` devant les guillemets, Python affiche les accolades telles quelles : `print("{nom}")` affiche `{nom}` et non `Lina`.

1.5 Instructions conditionnelles

Définition – `if` / `elif` / `else`

Une instruction conditionnelle exécute un bloc d'instructions **seulement si** une condition est vraie. `elif` (« sinon si ») permet d'enchaîner d'autres tests, `else` (« sinon ») traite tous les cas restants.

Propriété – Opérateurs de comparaison et logiques

— Comparaisons : `==` (égal), `!=` (différent), `<`, `<=`, `>`, `>=`.

— Opérateurs logiques : `and` (et), `or` (ou), `not` (non).

Le résultat d'une comparaison est un booléen : `3 < 5` vaut `True`.

Méthode – Afficher une appréciation selon une note

```
note = float(input("Note sur 20 : "))
if note >= 16:
    print("Tres bien")
elif note >= 12:
    print("Bien")
elif note >= 10:
    print("Passable")
else:
    print("Insuffisant")
```

Les tests sont effectués **dans l'ordre** : dès qu'une condition est vraie, son bloc est exécuté et les suivants sont ignorés.

Attention

Ne pas oublier les **deux-points** `:` à la fin des lignes `if`, `elif` et `else`.

1.6 Boucles bornées et non bornées

1.6.1 A – Boucle bornée `for`

Définition – Boucle bornée

Une **boucle bornée** répète un bloc d'instructions un nombre de fois **connu à l'avance**. `for i in range(n):` répète le bloc `n` fois, avec `i` prenant les valeurs `0, 1, ..., n-1`.

Propriété – La fonction `range`

— `range(5)` : 0, 1, 2, 3, 4 ;

— `range(2, 6)` : 2, 3, 4, 5 (la borne de fin est **exclue**) ;

— `range(0, 10, 2)` : 0, 2, 4, 6, 8 (pas de 2).

On peut aussi parcourir une chaîne : `for lettre in "SNT":`.

Méthode – Calculer une somme avec un accumulateur

```
# Somme des entiers de 1 a 100
somme = 0
for i in range(1, 101):
    somme = somme + i
print(f"Somme : {somme}")    # affiche : Somme : 5050
```

1.6.2 B – Boucle non bornée `while`

Définition – Boucle non bornée

Une **boucle non bornée** répète un bloc **tant qu'**une condition reste vraie : on ne connaît pas à l'avance le nombre de répétitions.

Méthode – Nombre d'années pour doubler une épargne

```
# Un capital de 1000 euros place a 5 % par an
capital = 1000
annees = 0
while capital < 2000:
    capital = capital * 1.05
    annees = annees + 1
print(f"Il faut {annees} ans")    # affiche : Il faut 15 ans
```

Attention

Si la condition d'un `while` ne devient jamais fausse, le programme tourne indéfiniment (**boucle infinie**). Il faut s'assurer qu'une variable de la condition est modifiée dans la boucle.

1.7 Fonctions

Définition – Fonction

Une **fonction** est un bloc d'instructions auquel on donne un nom, qui peut recevoir des **paramètres** et **renvoyer** un résultat avec `return`. On la **définit** une fois avec `def`, puis on peut l'**appeler** autant de fois que nécessaire.

Méthode – Définir puis appeler une fonction

```
def imc(masse, taille):
    """Renvoie l'indice de masse corporelle."""
    return masse / taille ** 2

resultat = imc(60, 1.70)        # appel de la fonction
print(f"IMC : {resultat:.1f}")  # affiche : IMC : 20.8
```

Propriété – Intérêt des fonctions

— Éviter de recopier le même code plusieurs fois.
— Découper un problème complexe en petites tâches simples.
— Rendre le programme plus lisible et plus facile à corriger.

Attention

`return` renvoie une valeur réutilisable dans la suite du programme, alors que `print` se contente de l'afficher. Une fonction s'arrête dès qu'elle exécute un `return`.

Méthode – Une fonction qui renvoie un booléen

```
def est_majeur(age):
    return age >= 18

print(est_majeur(15))   # False
print(est_majeur(20))   # True
```

1.8 Ateliers Python

Les ateliers se font sur ordinateur, dans un éditeur Python ou directement sur https://utspe.com/snt/chapitre-1. Pour chacun : recopier le programme de départ, suivre les étapes dans l'ordre et exécuter le programme après chaque modification.

Atelier 1 – Mon forfait mobile variables, calculs, f-strings

Objectif : Utiliser des variables pour calculer le coût d'un forfait mobile et afficher un bilan propre avec une f-string.

1. Exécuter le programme et lire ce qu'il affiche.
2. Compléter le calcul de `cout_annuel` (12 mois d'abonnement).
3. Calculer `prix_par_go` : le prix mensuel divisé par le nombre de Go.
4. Afficher les deux résultats avec une f-string, arrondis à 2 décimales (`{x:.2f}`).

5. Changer les valeurs (forfait à 15,99 € pour 200 Go) et comparer.

Programme de départ :

```
prix_mensuel = 9.99       # en euros
data_go = 100             # Go inclus par mois

cout_annuel = 0           # a completer
prix_par_go = 0           # a completer

print("Forfait :", prix_mensuel, "euros par mois")
# a completer : afficher cout_annuel et prix_par_go avec une f-string
```

Coup de pouce : Le coût annuel vaut `prix_mensuel * 12`. Dans une f-string, on écrit `f"Coût annuel : {cout_annuel:.2f} €"`.

Corrigé rédigé

```
prix_mensuel = 9.99       # en euros
data_go = 100             # Go inclus par mois

cout_annuel = prix_mensuel * 12
prix_par_go = prix_mensuel / data_go

print("Forfait :", prix_mensuel, "euros par mois")
print(f"Cout annuel : {cout_annuel:.2f} euros")
print(f"Prix d'un Go : {prix_par_go:.2f} euros")
```

Atelier 2 – Vérificateur de mot de passe

chaînes, len(), if / elif / else

Objectif : Évaluer la robustesse d'un mot de passe selon sa longueur, puis selon la présence d'un chiffre.

1. La fonction `len(mdp)` donne le nombre de caractères. Afficher la longueur de `mdp`.
2. Afficher « faible » si la longueur est inférieure à 8, « moyen » entre 8 et 11, « robuste » à partir de 12.
3. Tester avec `"soleil"`, `"soleil2026"` et `"Soleil-Levant-2026"`.
4. Bonus : avec une boucle `for c in mdp:`, vérifier si le mot de passe contient au moins un chiffre.

Programme de départ :

```
mdp = "soleil2026"

longueur = len(mdp)
print(f"Le mot de passe contient {longueur} caracteres")

# a completer : if / elif / else
```

Coup de pouce : On teste d'abord `if longueur < 8:`, puis `elif longueur < 12:`, et enfin `else:`. Pour le bonus, le test `c in "0123456789"` vaut `True` si `c` est un chiffre.

Corrigé rédigé

```
mdp = "soleil2026"

longueur = len(mdp)
print(f"Le mot de passe contient {longueur} caracteres")

```

```python
if longueur < 8:
    print("Mot de passe faible")
elif longueur < 12:
    print("Mot de passe moyen")
else:
    print("Mot de passe robuste")

# Bonus : presence d'un chiffre
contient_chiffre = False
for c in mdp:
    if c in "0123456789":
        contient_chiffre = True
print("Contient un chiffre :", contient_chiffre)
```

Atelier 3 – La croissance d'un compte sur un réseau social

boucles for et while

Objectif : Modéliser l'évolution du nombre d'abonnés d'un compte, qui gagne 15 % d'abonnés chaque semaine.

1. Avec une boucle `for`, afficher le nombre d'abonnés (arrondi à l'unité) pendant 8 semaines, en partant de 200.
2. Avec une boucle `while`, déterminer au bout de combien de semaines le compte dépasse 1 000 abonnés.
3. Modifier le taux (10 %, puis 25 %) et observer l'effet sur le nombre de semaines.

Programme de départ :

```python
abonnes = 200
for semaine in range(1, 9):
    abonnes = abonnes * 1.15
    print(f"Semaine {semaine} : {abonnes:.0f} abonnes")

# a completer : boucle while pour depasser 1000 abonnes
```

Coup de pouce : Repartir de `abonnes = 200` et `semaine = 0`, puis répéter `while abonnes <= 1000:` en augmentant `abonnes` de 15 % et `semaine` de 1.

Corrigé rédigé

```python
abonnes = 200
for semaine in range(1, 9):
    abonnes = abonnes * 1.15
    print(f"Semaine {semaine} : {abonnes:.0f} abonnes")

abonnes = 200
semaine = 0
while abonnes <= 1000:
    abonnes = abonnes * 1.15
    semaine = semaine + 1
print(f"Plus de 1000 abonnes au bout de {semaine} semaines")
```

Atelier 4 – Le poids d'une photo numérique

fonctions, return, boucle for

Objectif : Écrire une fonction qui calcule la taille d'une image non compressée (thème « Photographie numérique ») : chaque pixel est codé sur 3 octets (rouge, vert, bleu).

1. Compléter la fonction `taille_image(largeur, hauteur)` pour qu'elle renvoie la taille en mégaoctets (1 Mo = 1 000 000 octets).
2. Tester la fonction sur une image de 1920 × 1080 pixels.
3. Avec une boucle, afficher la taille des images carrées de 1000 × 1000 à 5000 × 5000 pixels, de 1000 en 1000.

Programme de départ :

```
def taille_image(largeur, hauteur):
    """Renvoie la taille en Mo d'une image non compressee."""
    octets = 0        # a completer : 3 octets par pixel
    return octets / 1000000

print(f"Full HD : {taille_image(1920, 1080):.1f} Mo")
```

Coup de pouce : Le nombre de pixels vaut `largeur * hauteur` ; il faut le multiplier par 3. Pour la boucle : `for cote in range(1000, 6000, 1000):`.

Corrigé rédigé

```
def taille_image(largeur, hauteur):
    """Renvoie la taille en Mo d'une image non compressee."""
    octets = largeur * hauteur * 3
    return octets / 1000000

print(f"Full HD : {taille_image(1920, 1080):.1f} Mo")

for cote in range(1000, 6000, 1000):
    print(f"{cote} x {cote} pixels : {taille_image(cote, cote):.1f} Mo")
```

Atelier 5 – Mini-projet : le code de César

fonctions, chaînes, boucles, conditions

Objectif : Chiffrer un message en décalant chaque lettre de `k` rangs dans l'alphabet (A → D pour k = 3), comme Jules César.

1. La fonction `ord("A")` donne le code de la lettre (65) et `chr(65)` redonne `"A"`. Tester `ord("C")` et `chr(70)`.
2. Compléter `decaler(lettre, k)` : le rang de la lettre vaut `ord(lettre) - 65` (de 0 à 25) ; on lui ajoute `k` et on prend le reste par 26.
3. Écrire `chiffrer(message, k)` qui décale chaque lettre et laisse les espaces inchangés.
4. Chiffrer `"RENDEZ VOUS EN SALLE INFO"` avec k = 3, puis déchiffrer avec k = −3.

Programme de départ :

```
def decaler(lettre, k):
    rang = ord(lettre) - 65
    nouveau_rang = rang        # a completer
    return chr(nouveau_rang + 65)

print(decaler("A", 3))   # doit afficher D
print(decaler("Y", 3))   # doit afficher B
```

Coup de pouce : `nouveau_rang = (rang + k) % 26`. Dans `chiffrer`, on part d'une chaîne vide `resultat = ""` et on ajoute les caractères un par un : `resultat = resultat + ...`.

Corrigé rédigé

```python
def decaler(lettre, k):
    rang = ord(lettre) - 65
    nouveau_rang = (rang + k) % 26
    return chr(nouveau_rang + 65)

def chiffrer(message, k):
    resultat = ""
    for c in message:
        if c == " ":
            resultat = resultat + " "
        else:
            resultat = resultat + decaler(c, k)
    return resultat

secret = chiffrer("RENDEZ VOUS EN SALLE INFO", 3)
print(secret)
print(chiffrer(secret, -3))
```

1.9 Exercices progressifs

Exercice

Exercice 1 – Tableau d'état. Donner la valeur de `x` et `y` à la fin du programme :

```python
x = 4
y = 10
x = y - x
y = y // x
x = x ** 2
```

Corrigé rédigé

Ligne 3 : `x = 10 - 4 = 6`. Ligne 4 : `y = 10 // 6 = 1` (quotient entier). Ligne 5 : `x = 6 ** 2 = 36`.
À la fin : `x` vaut **36** et `y` vaut **1**.

Exercice

Exercice 2 – Types. Donner le type de chaque valeur : `7`, `7.0`, `"7"`, `7 > 3`, `7 / 2`, `7 // 2`.

Corrigé rédigé

`7` : `int` ; `7.0` : `float` ; `"7"` : `str` ; `7 > 3` : `bool` (vaut `True`) ; `7 / 2` : `float` (vaut `3.5`) ; `7 // 2` : `int` (vaut `3`).

Exercice

Exercice 3 – Pair ou impair. Écrire un programme qui demande un entier à l'utilisateur et affiche « pair » ou « impair ». *Indice : utiliser le reste `%`.*

Corrigé rédigé

Un entier est pair lorsque le reste de sa division par 2 est nul.

```
n = int(input("Entier : "))
if n % 2 == 0:
    print(f"{n} est pair")
else:
    print(f"{n} est impair")
```

Exercice

Exercice 4 – Table de multiplication. Écrire un programme qui demande un entier `n` puis affiche sa table de multiplication de `n x 1` à `n x 10`, une ligne par calcul, sous la forme `7 x 3 = 21`. *Indice : utiliser une boucle `for` et une f-string.*

Corrigé rédigé

On connaît le nombre de répétitions (10) : on utilise une boucle bornée, avec `i` allant de 1 à 10.

```
n = int(input("n = "))
for i in range(1, 11):
    print(f"{n} x {i} = {n * i}")
```

Exercice

Exercice 5 – Population. Une ville compte 20 000 habitants et sa population augmente de 3 % par an. Écrire un programme avec une boucle `while` qui détermine au bout de combien d'années la population dépassera 30 000 habitants.

Corrigé rédigé

Augmenter de 3 % revient à multiplier par 1,03. On répète tant que la population ne dépasse pas 30 000.

```
population = 20000
annees = 0
while population <= 30000:
    population = population * 1.03
    annees = annees + 1
print(f"Au bout de {annees} ans : {population:.0f} habitants")
```

Le programme affiche `Au bout de 14 ans : 30252 habitants`.

Exercice

Exercice 6 – Fonctions et données.

1. Écrire une fonction `conversion(octets)` qui renvoie la taille en kilo-octets (1 ko = 1 000 octets).
2. Écrire une fonction `compte_voyelles(mot)` qui renvoie le nombre de voyelles d'une chaîne de caractères.
3. Tester ces deux fonctions et afficher les résultats avec une f-string, par exemple : `250000 octets = 250.0 ko` et `internet contient 3 voyelles`.

Corrigé rédigé

```python
def conversion(octets):
    return octets / 1000

def compte_voyelles(mot):
    total = 0
    for lettre in mot:
        if lettre in "aeiouy":
            total = total + 1
    return total

print(f"250000 octets = {conversion(250000)} ko")
print(f"internet contient {compte_voyelles('internet')} voyelles")
```

1.10 Exercices d'approfondissement

Exercice

Exercice 7 – Suivre une boucle à la main. Compléter un tableau donnant les valeurs de `i` et de `s` à chaque tour de boucle, puis donner l'affichage final.

```python
s = 0
for i in range(1, 5):
    s = s + i * i
print(s)
```

Corrigé rédigé

Tour 1 : `i = 1`, `s = 1` ; tour 2 : `i = 2`, `s = 5` ; tour 3 : `i = 3`, `s = 14` ; tour 4 : `i = 4`, `s = 30`. `range(1, 5)` s'arrête avant 5 : le programme affiche `30` (la somme $1^2 + 2^2 + 3^2 + 4^2$).

Exercice

Exercice 8 – Chasse aux erreurs. Ce programme doit afficher « Majeur » ou « Mineur » puis l'âge dans un an. Il contient **quatre erreurs**. Les trouver et les corriger.

```python
age = input("Ton age ? ")
if age = 18
print("Majeur")
else:
    print("Mineur")
print("Dans un an tu auras " + age + 1 + " ans")
```

Corrigé rédigé

1. `input()` renvoie une chaîne : il faut `age = int(input("Ton age ? "))`.
2. Le test s'écrit `age >= 18` (et pas `=`, qui est une affectation).
3. Il manque les deux-points `:` à la fin de la ligne `if`.
4. `print("Majeur")` doit être indenté.

Enfin, on ne peut pas additionner une chaîne et un nombre : une f-string règle le problème.

```python
age = int(input("Ton age ? "))
if age >= 18:
    print("Majeur")
else:
    print("Mineur")
print(f"Dans un an tu auras {age + 1} ans")
```

Exercice

Exercice 9 – Temps de téléchargement. Thème « Internet ». Le débit d'une connexion s'exprime en mégabits par seconde (Mb/s) et 1 octet = 8 bits.
1. Écrire une fonction `temps_telechargement(taille_mo, debit)` qui renvoie la durée en secondes pour télécharger un fichier de `taille_mo` mégaoctets.
2. Calculer la durée pour un fichier de 700 Mo avec un débit de 20 Mb/s.
3. Afficher cette durée en minutes et secondes avec `//` et `%`, sous la forme `4 min 40 s`.

Corrigé rédigé

700 Mo = $700 \times 8 = 5\,600$ Mb, et $5\,600 \div 20 = 280$ s. Puis `280 // 60` = 4 minutes et `280 % 60` = 40 secondes.

```python
def temps_telechargement(taille_mo, debit):
    return taille_mo * 8 / debit

duree = int(temps_telechargement(700, 20))
print(f"Duree : {duree} s")
print(f"Soit {duree // 60} min {duree % 60} s")
```

Exercice

Exercice 10 – Table de conversion de températures. Thème « Données structurées ». Une station météo connectée mesure en degrés Celsius ; un site américain affiche des degrés Fahrenheit, avec $F = 1{,}8 \times C + 32$.
1. Écrire une fonction `fahrenheit(c)`.
2. Avec `for c in range(-10, 41, 10):`, afficher la table de conversion de -10 °C à 40 °C.

Corrigé rédigé

`range(-10, 41, 10)` donne -10, 0, 10, 20, 30, 40 : le troisième argument est le **pas**.

```python
def fahrenheit(c):
    return 1.8 * c + 32

for c in range(-10, 41, 10):
    print(f"{c} C = {fahrenheit(c):.0f} F")
```

Exercice

Exercice 11 – Années bissextiles. Une année est bissextile si elle est divisible par 4 mais pas par 100, ou si elle est divisible par 400.
1. Écrire une fonction `est_bissextile(annee)` qui renvoie `True` ou `False`, avec les opérateurs `and`, `or` et `%`.

2. Vérifier : 2024 → True, 1900 → False, 2000 → True.
3. Compter les années bissextiles de 2000 à 2100 (incluses).

Corrigé rédigé

« Divisible par 4 » se traduit par `annee % 4 == 0`. Entre 2000 et 2100, on trouve **25** années bissextiles (2100 ne l'est pas).

```
def est_bissextile(annee):
    return (annee % 4 == 0 and annee % 100 != 0) or annee % 400 == 0

print(est_bissextile(2024), est_bissextile(1900), est_bissextile(2000))

compteur = 0
for annee in range(2000, 2101):
    if est_bissextile(annee):
        compteur = compteur + 1
print(f"{compteur} annees bissextiles de 2000 a 2100")
```

Exercice

Exercice 12 – Défi : le nombre mystère. L'ordinateur choisit un entier au hasard entre 1 et 100 avec `random.randint(1, 100)` (après `import random`). Le joueur propose des nombres et l'ordinateur répond « trop grand » ou « trop petit » jusqu'à ce qu'il trouve. À la fin, afficher le nombre d'essais.
Questions : quelle boucle faut-il utiliser ? Pourquoi ? Quelle est la meilleure stratégie pour gagner en au plus 7 essais ?

Corrigé rédigé

On ne sait pas à l'avance combien d'essais seront nécessaires : on utilise une boucle `while`. La meilleure stratégie est la **dichotomie** : proposer le milieu de l'intervalle restant (50, puis 25 ou 75…). L'intervalle est divisé par 2 à chaque essai, et $2^7 = 128 > 100$: 7 essais suffisent toujours.

```
import random

mystere = random.randint(1, 100)
essais = 0
proposition = 0
while proposition != mystere:
    proposition = int(input("Ta proposition : "))
    essais = essais + 1
    if proposition > mystere:
        print("Trop grand")
    elif proposition < mystere:
        print("Trop petit")
print(f"Bravo, trouve en {essais} essais !")
```

1.11 QCM – Teste tes connaissances

Pour chaque question, une seule réponse est correcte.

Q1. Que affiche `print(7 // 2)` ?

a) `3.5` **b)** `3` **c)** `1` **d)** `4`

Q2. Quel est le type de la valeur `"15"` ?

a) `int` **b)** `float` **c)** `str` **d)** `bool`

Q3. Que vaut `a` à la fin de ce programme ?

```
a = 3
a = a * 2 + 1
```

a) `3` **b)** `6` **c)** `7` **d)** `9`

Q4. La fonction `input()` renvoie...

a) toujours un entier **b)** toujours une chaîne de caractères **c)** le type de ce qui est saisi **d)** rien du tout

Q5. Qu'affiche ce programme ?

```
x = 2.5
print(f"x vaut {x * 2}")
```

a) `x vaut {x * 2}` **b)** `x vaut 5` **c)** `x vaut 5.0` **d)** une erreur

Q6. Quel test vérifie que l'entier `n` est pair ?

a) `n / 2 == 0` **b)** `n % 2 == 0` **c)** `n // 2 == 0` **d)** `n = 2`

Q7. Combien de fois « bonjour » est-il affiché ?

```
for i in range(2, 7):
    print("bonjour")
```

a) `4` **b)** `5` **c)** `6` **d)** `7`

Q8. Quelle est la dernière valeur affichée ?

```
for i in range(4):
    print(i)
```

a) `3` **b)** `4` **c)** `5` **d)** `0`

Q9. Qu'affiche ce programme ?

```
n = 1
while n < 50:
    n = n * 3
print(n)
```

a) `27` **b)** `50` **c)** `81` **d)** `243`

Q10. Qu'affiche ce programme ?

```
note = 12
if note >= 16:
    print("Tres bien")
elif note >= 12:
    print("Bien")
elif note >= 10:
    print("Assez bien")
else:
    print("Insuffisant")
```

a) `Tres bien` **b)** `Bien` **c)** `Bien puis Assez bien` **d)** `Insuffisant`

Q11. Qu'affiche ce programme ?

```
def f(x):
    return 2 * x + 1

print(f(f(1)))
```

a) `3` **b)** `5` **c)** `7` **d)** `9`

Q12. Qu'affiche `print("SNT" * 2 + "!")` ?

a) `SNT2!` **b)** `SNTSNT!` **c)** `SNT!SNT!` **d)** une erreur

Corrigé rédigé

Réponses du QCM.

Q1. **b)** `//` donne le quotient entier : $7 = 2 \times 3 + 1$.

Q2. **c)** Les guillemets en font une chaîne de caractères (`str`).

Q3. **c)** On calcule $3 \times 2 + 1 = 7$, puis on range 7 dans `a`.

Q4. **b)** Il faut convertir avec `int()` ou `float()` pour faire des calculs.

Q5. **c)** L'expression entre accolades est calculée ; 2.5×2 est un flottant, d'où `5.0`.

Q6. **b)** `n % 2` est le reste de la division par 2 : il vaut 0 pour un nombre pair.

Q7. **b)** `i` prend les valeurs 2, 3, 4, 5, 6 : cinq tours.

Q8. **a)** `range(4)` donne 0, 1, 2, 3 : on s'arrête avant 4.

Q9. **c)** n prend les valeurs 1, 3, 9, 27, 81 ; la boucle s'arrête dès que le test `n < 50` est faux, c'est-à-dire dès que `n >= 50`.

Q10. **b)** Seul le premier bloc dont la condition est vraie est exécuté.

Q11. **c)** f(1) = 3, puis f(3) = 7.

Q12. **b)** `*` répète la chaîne, puis `+` la concatène avec « ! ».

1.12 Fiche de synthèse

Fiche de synthèse

— **Variables et types :** affectation `x = valeur`. Types `int`, `float`, `str`, `bool`. Conversions `int()`, `float()`, `str()`.

— **Entrées / sorties :** `print()` affiche ; `input()` lit une saisie et renvoie toujours une chaîne.

— **F-strings :** `print(f"Bonjour {prenom}, tu as {age} ans")` ; `{x:.2f}` arrondit l'affichage à 2 décimales.

— **Conditions :** `if condition:` … `elif autre:` … `else:` … – tester l'égalité avec `==`, ne pas oublier les `:` et l'indentation.

— **Boucles :** `for i in range(n):` quand on connaît le nombre de répétitions ; `while condition:` quand on répète « tant que ». Attention aux boucles infinies.

— **Fonctions :** `def nom(parametres):` puis `return resultat`. On définit une fois, on appelle autant de fois que nécessaire.