🦸 Superhéros : couleurs, bonus et combat

Tu ajoutes trois nouvelles fonctionnalités à ton générateur de superhéros : des couleurs dans la console avec Rich, un bonus de caractéristique lancé au dé, et un combat final contre un boss.

Exercice

Ces étapes complètent ton programme de superhéros. Tu pars du code que tu as déjà écrit (origine, caractéristiques `force`, `agilite`, `intelligence`, `endurance`…) et tu lui ajoutes de la couleur, un bonus et un combat.

🎯 Objectifs

1. Colorier les affichages importants de la console avec le module Rich.
2. Appliquer un bonus aléatoire à une caractéristique selon l'origine du héros.
3. Résoudre un combat contre un boss à l'aide d'une boucle `for` et d'une condition.

🎨 Avant tout : ajouter des couleurs avec Rich

Ce qu'on veut faire

On veut colorier certains textes dans la console pour rendre le programme plus lisible — sans changer la structure du code.

Installation dans Thonny

Rich n'est pas installé par défaut. Dans Thonny :

1. Menu **Outils** → **Gérer les paquets…** (ou **CTRL+T**)
2. Tape `rich` dans la barre de recherche
3. Clique sur **Installer**

L'import à ajouter

Ajoute cette ligne **tout en haut** du fichier, avec les autres imports :

```
from rich import print
```

> 💡 Cette seule ligne remplace silencieusement le `print()` de Python par celui de Rich. **Ton code existant ne change pas du tout.** La différence : tu peux maintenant ajouter des balises de couleur dans tes textes.

Comment colorier du texte

On entoure le texte à colorier avec des balises entre crochets :

```
print("[bold green]Victoire ![/bold green]")
print("[bold red]Défaite...[/bold red]")
print(f"Score : [yellow]{score}[/yellow]")
print("[dim]Texte discret[/dim]")
```

Les couleurs : `red` `green` `yellow` `cyan` `magenta` `blue` `white` Les styles : `bold` (gras) · `italic` · `dim` (atténué)

> ⚠️ **RÈGLE DE LISIBILITÉ :** colorie uniquement les valeurs importantes (résultats, erreurs, succès). Garde le texte descriptif en blanc normal. Trop de couleurs = aucune couleur.

ÉTAPE 5.5 — Bonus de caractéristique

Ce qu'on veut faire

Ton origine t'accorde un bonus sur une caractéristique. On lance un dé à 6 faces (d6) et on ajoute le résultat à la stat correspondante.

Chaque origine booste une stat différente :

Origine	Stat boostée
Mutation génétique	Force
Technologie	Agilité
Magie ancienne	Intelligence

Origine	Stat boostée
Radioactivité	Endurance

Quelle structure utiliser ?

On a besoin du **nom** de la stat boostée pour l'afficher → liste parallèle. On a besoin de **modifier la variable** → `if/elif` (comme à l'étape 5).

> 💡 Les deux listes `origines` et `boost_noms` doivent être définies au même endroit que les autres, **tout en haut du programme**.

Le code

```
# À ajouter en haut, avec les autres listes
boost_noms = ["Force", "Agilité", "Intelligence", "Endurance"]

# --- Étape 5.5 ---
# On récupère le nom de la stat boostée grâce à l'index du choix
stat_boostee = boost_noms[choix]

# On lance un d6
bonus = random.randint(1, 6)

# On applique le bonus à la bonne variable
if choix == 0:
    force += bonus
elif choix == 1:
    agilite += bonus
elif choix == 2:
    intelligence += bonus
elif choix == 3:
    endurance += bonus

print(f"[bold yellow] Bonus d6 : +{bonus} en {stat_boostee}[/bold yellow]")
print(f"[dim]{stat_boostee} passe à {intelligence}[/dim]")
```

> 💡 `force += bonus` est une écriture courte pour `force = force + bonus` .

Exemple d'exécution

```
 Bonus d6 : +4 en Intelligence     ← en jaune gras
Intelligence passe à 22            ← en gris discret
```

ÉTAPE 8 — Combat final

Ce qu'on veut faire

Ton héros affronte un boss. Le combat se résout en deux étapes :

1. On calcule le **nombre de coups** = `force + intelligence`
2. On lance autant de **d6** que de coups — la somme représente les dégâts infligés
3. Si les dégâts dépassent les PV du boss → victoire, sinon défaite

Quelle structure utiliser ?

On répète le même lancer un certain nombre de fois. ➡️ Le nombre de répétitions est connu à l'avance → boucle `for`.

Pour la victoire ou la défaite → résultat différent selon une condition → `if/else`.

Formule des PV du boss

Le boss doit être **difficile mais pas impossible** à battre. La formule choisie garantit environ **50 % de chances de victoire** :

```
boss_pv = nb_coups * 3 + random.randint(0, nb_coups)
```

> 🧠 **POURQUOI ÇA MARCHE ?**
>
> Chaque d6 donne en moyenne **3.5**. Donc la somme des lancers tourne autour de `nb_coups × 3.5`. Le boss a entre `nb_coups × 3` et `nb_coups × 4` PV — ce qui le place juste autour de cette moyenne. Résultat : une chance sur deux de gagner, quelle que soit la valeur de `nb_coups`.

Le code

```
nb_coups = force + intelligence
boss_pv  = nb_coups * 3 + random.randint(0, nb_coups)

print(f"\n[bold red]👹 Boss — Points de vie : {boss_pv}[/bold red]")
print(f"Tu lances [white]{nb_coups}[/white] dés (force {force} + intelligence {intelligence})")

# Lancer nb_coups dés et additionner les résultats
total_degats = 0
for i in range(nb_coups):
    total_degats += random.randint(1, 6)

```

```python
print(f"Dégâts infligés : [bold white]{total_degats}[/bold white]")

# Résultat du combat
if total_degats > boss_pv:
    print(f"[bold green]✓ Combat GAGNÉ ! ({total_degats} > {boss_pv})[/bold green]")
else:
    print(f"[bold red]✗ Combat PERDU. ({total_degats} ≤ {boss_pv})[/bold red]")
```

Exemple d'exécution

```
👹 Boss — Points de vie : 78      ← en rouge gras
Tu lances 24 dés (force 9 + intelligence 15)
Dégâts infligés : 83
✓ Combat GAGNÉ ! (83 > 78)        ← en vert gras
```

🚀 Bonus — Tournoi de plusieurs combats

Au lieu d'un seul combat, ton héros dispute **3 combats** contre des boss différents. On compte ses victoires.

```python
NB_COMBATS = 3
victoires  = 0

for combat in range(NB_COMBATS):
    # Chaque boss a ses propres PV
    pv_boss = nb_coups * 3 + random.randint(0, nb_coups)
    degats  = sum(random.randint(1, 6) for _ in range(nb_coups))

    if degats > pv_boss:
        victoires += 1
        print(f"  Combat {combat + 1} : [green]Victoire[/green] ({degats} vs {pv_boss})")
    else:
        print(f"  Combat {combat + 1} : [red]Défaite[/red]  ({degats} vs {pv_boss})")

print(f"\n[bold]Résultat final : {victoires}/{NB_COMBATS} victoires[/bold]")
```

> 💡 `sum(... for _ in range(n))` est une façon compacte de lancer `n` dés et d'additionner les résultats d'un coup. C'est équivalent à la boucle `for` ci-dessus — à toi de choisir celle que tu préfères.

Exemple d'exécution

```
  Combat 1 : Victoire (88 vs 81)    ← Victoire en vert
  Combat 2 : Défaite  (71 vs 79)    ← Défaite en rouge
```

```
  Combat 3 : Victoire (92 vs 85)    ← Victoire en vert

Résultat final : 2/3 victoires
```

✍️ À retenir

- Le module **Rich** colore tes affichages sans changer la logique de ton code.
- Une **liste parallèle** (`boost_noms`) sert à retrouver un texte à partir d'un index.
- `+=` est un raccourci pour ajouter une valeur à une variable existante.
- Une **boucle** `for` répète un lancer un nombre de fois connu à l'avance.
- Un `if/else` choisit entre deux issues (victoire / défaite).