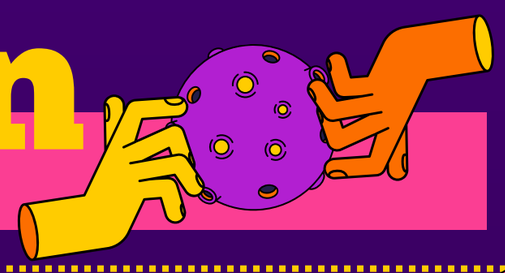
Python



Commandes de base

- `print(objet)` Affiche un objet
- `input(prompt)` Demande une saisie utilisateur
- `type(objet)` Renvoie le type d'un objet
- `id(objet)` Renvoie l'identifiant unique d'un objet
- `help(objet)` Affiche l'aide d'un objet
- `dir(objet)` Liste les attributs et méthodes disponibles pour un objet
- `len(objet)` Renvoie la longueur d'un objet

Définitions

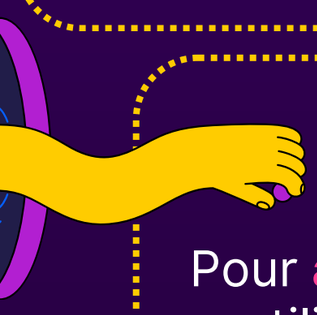
- Chaîne de caractère `string` ➡ (“3”+”5"+”Sal”+”ut”) = 35Salut
- Nombres entiers `int` ➡ (3+2+1) = 6
- Nombres à virgule flottante `float` ➡ (3.02+5.05) = 8,07
- Nombres complexes `complex` ➡ (1+4j+8j+2) = 3+12j

les principaux opérateurs de comparaison

`==`	Égalité	`x == x` ➡ True	`>`	Supérieur	`x > x` ➡ False
`!=`	Différent	`x != x` ➡ False	`<=`	Inférieur ou égal	`x <= x` ➡ True
`<`	Inférieur	`x < x` ➡ False	`>=`	Supérieur ou égal	`x >= x` ➡ True

les principaux opérateurs arithmétiques

`x + y`	la somme de x et y (x+y)	`x ** y`	x est élever à la puissance de y (x^y)
`x - y`	la différence entre x et y (x-y)	`x / y`	le quotient de x et y (x / y)
`x * y`	le produit de x et y (x*y)	`x % y`	le reste de x divisé par y



f-string

Pour **afficher** des **variables**, la **f-string** est souvent utilisée car elle permet d'**insérer** facilement les **variables** dans la **chaîne de caractères** à afficher

```python
nom = "Dupont"
prenom = "Jean"
age = 30
print(f"Bonjour, je m'appelle {prenom} {nom} et j'ai {age} ans.")
```

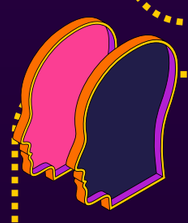
booléens

Type de **donnée** qui peut prendre **deux** valeurs

`True` Vrai `False` Faux

utilisés pour **représenter** des **états logiques** ou des **résultats de comparaisons**

```python
age = 18
a_le_droit_de_voter = age >= 18  # True, car 18 est supérieur ou égal à 
if a_le_droit_de_voter:
    print("Vous pouvez voter.")
else:
    print("Vous ne pouvez pas voter.")
```



instructions if/else

L'instruction **if** teste une **condition**. Si elle est **True**, le code associé est **exécuté**

L'instruction **else** permet de définir un bloc à exécuter lorsque la **condition** if est **False**

L'instruction **elif** permet de **tester** plusieurs **conditions**

```python
ensoleille = False
neige = True
if ensoleille:
    print("on va à la plage !")
elif neige:
    print("on fait un bonhomme de neige")
else:
    print("on reste à la maison !")
```

les principaux opérateurs logiques

and Combine deux conditions ; les deux doivent être vraies pour que le résultat soit vrai

```python
age = 20
revenu = 3000

if age >= 18 and revenu > 2000:
    print("Condition remplie")  # Condition remplie
else:
    print("Condition non remplie")
```

or Combine deux conditions ; au moins une des deux doit être vraie pour que le résultat soit vrai

```python
temperature = 35
humidité = 60

if temperature > 30 or humidité > 70:
    print("Il fait chaud ou humide")  # Il fait chaud ou humide
else:
    print("Temps agréable")
```

not Inverse la valeur de vérité d'une condition. Si elle est vraie, elle devient fausse, et vice versa

```python
is_admin = False

if not is_admin:
    print("Accès refusé")  # Accès refusé
else:
    print("Bienvenue administrateur")
```

in Vérifie si un élément appartient à une liste, un tuple, ou un autre conteneur

```python
fruit = "pomme"
fruits_autorisés = ["pomme", "banane", "orange"]

if fruit in fruits_autorisés:
    print("Fruit autorisé")  # Fruit autorisé
else:
    print("Fruit non autorisé")
```

not in Vérifie si un élément n'est pas présent dans une liste, un tuple, ou un autre conteneur

```python
jour = "dimanche"
jours_travail = ["lundi", "mardi", "mercredi", "jeudi", "vendredi"]

if jour not in jours_travail:
    print("C'est un jour de repos")  # C'est un jour de repos
else:
    print("C'est un jour de travail")
```

les principales structures de données

list

Collections ordonnées et modifiables

```python
# Création d'une liste
ma_liste = [1, 2, 3, 4, 5]

# Une liste avec des types mixtes
liste_mixte = [42, "Python", 3.14, True]

# Accès aux éléments
print(ma_liste[0])  # Affiche 1 (1er élément)
print(ma_liste[-1])  # Affiche 5 (dernier élément)

# Modification
ma_liste[2] = 10  # Change le 3e élément
```

set

collections non ordonnées et uniques

```python
# Création d'un ensemble
mon_ensemble = {1, 2, 3, 2, 4}
print(mon_ensemble)  # Affiche {1, 2, 3, 4} (les doublons sont éliminés)

# Ajouter ou supprimer des éléments
mon_ensemble.add(5)
mon_ensemble.remove(3)
```

dict

collections ordonnées

```python
# Création d'un dictionnaire
mon_dico = {
    "nom": "Alice",
    "âge": 25,
    "ville": "Paris"
}

# Accès à une valeur
print(mon_dico["nom"])  # Affiche "Alice"

# Ajout ou modification
mon_dico["profession"] = "Développeuse"
```

tuple

Collections ordonnées mais immuables

```python
# Création d'un tuple
mon_tuple = (1, 2, 3)

# Accès aux éléments
print(mon_tuple[0])  # Affiche 1
```

str

Collections de caractères immuables

```python
# Création d'une chaîne
ma_chaine = "Bonjour tout le monde"

# Une chaîne contenant des caractères spéciaux
chaine_speciale = "Python\nC'est génial !\t🚀"

# Accès aux caractères
print(ma_chaine[0])  # Affiche 'B' (1er caractère)
print(ma_chaine[-1])  # Affiche 'e' (dernier caractère)
```

Python

Principales commandes pour modifier une liste (list)

append() Ajouter un élément à la fin de la liste

```
ma_liste = [1, 2, 3]
ma_liste.append(4)
print(ma_liste)  # [1, 2, 3, 4]
```

extend() Ajoute plusieurs éléments à la fin de la liste

```
ma_liste = [1, 2, 3]
ma_liste.extend([4, 5])
print(ma_liste)  # [1, 2, 3, 4, 5]
```

insert() Ajouter un élément à une position donnée

```
ma_liste = [1, 2, 3]
ma_liste.insert(1, 99)  # Insère 99 à l'indice 1
print(ma_liste)  # [1, 99, 2, 3]
```

remove() Supprime la première occurrence d'une valeur

```
ma_liste = [1, 2, 3, 2]
ma_liste.remove(2)  # Supprime le premier 2
print(ma liste)  # [1, 3, 2]
```

pop() Supprime et renvoie l'élément à un indice donné

```
ma_liste = [1, 2, 3]
element = ma_liste.pop(1)  # Supprime l'élément à l'indice 1
print(ma_liste)  # [1, 3]
print(element)  # 2
```

clear() Vide complètement la liste

```
ma_liste = [1, 2, 3]
ma_liste.clear()
print(ma_liste)  # []
```

del Supprimer un élément à un indice ou la liste entière

```
ma_liste = [1, 2, 3]
del ma_liste[1]  # Supprime l'élément à l'indice 1
print(ma_liste)  # [1, 3]

# Supprimer toute la liste
del ma_liste
```

sort() Trie la liste

```
ma_liste = [4, 2, 9, 1]
ma_liste.sort()  # Trie par ordre croissant
print(ma_liste)  # [1, 2, 4, 9]
```

```
ma_liste = [4, 2, 9, 1]
ma_liste.sort(reverse=True)  # Trie par ordre décroissant
print(ma_liste)  # [9, 4, 2, 1]
```

```
ma_liste = ["chat", "chien", "aigle", "abeille"]
ma_liste.sort(key=len)  # Trie par longueur des chaînes
print(ma_liste)  # ['chat', 'chien', 'aigle', 'abeille']
```

reverse() Inverse l'ordre des éléments dans la liste

```
ma_liste = [1, 2, 3]
ma_liste.reverse()
print(ma_liste)  # [3, 2, 1]
```

count() Compte le nombre de fois où une valeur apparaît

```
ma_liste = [10, 20, 30, 20, 20, 40]
compte = ma_liste.count(20)  # Compte le nombre de fois que 20 apparaît
print(compte)  # Résultat : 3
```

index() Renvoie l'indice de la première occurrence de la valeur

```
ma_liste = [10, 20, 30, 40, 20]
indice = ma_liste.index(20, 2)  # Cherche 20 à partir de l'indice 2
print(indice)  # Résultat : 4 (car le 20 suivant est à l'indice 4)

# Exemple avec `end`
indice = ma_liste.index(20, 1, 3)  # Cherche 20 entre les indices 1 et 3
print(indice)  # Résultat : 1
```

Principales commandes pour modifier un dictionnaire (dict)

get(key, [default]) Renvoie la valeur d'une clé ou une valeur par défaut si elle n'existe pas

```
mon_dict = {"nom": "Alice", "age": 25}
print(mon_dict.get("nom"))  # Alice
print(mon_dict.get("profession", "Inconnu"))  # Inconnu
```

keys() Renvoie toutes les clés du dictionnaire sous forme de vue

```
mon_dict = {"nom": "Alice", "age": 25}
print(mon_dict.keys())  # dict_keys(['nom', 'age'])
```

values() Renvoie toutes les valeurs du dictionnaire sous forme de vue

```
mon_dict = {"nom": "Alice", "age": 25}
print(mon_dict.values())  # dict_values(['Alice', 25])
```

items() Renvoie toutes les paires clé-valeur sous forme de tuples

```
mon_dict = {"nom": "Alice", "age": 25}
print(mon_dict.items())  # dict_items([('nom', 'Alice'), ('age', 25)])
```

update(other_dict) Met à jour un dictionnaire avec un autre dictionnaire

```
mon_dict = {"nom": "Alice", "age": 25}
mon_dict.update({"profession": "Ingénieur"})
print(mon_dict)  # {'nom': 'Alice', 'age': 25, 'profession': 'Ingénieur'}
```

pop(key, [default]) Supprime et renvoie la valeur associée à une clé. Si la clé n'existe pas, renvoie la valeur par défaut

```
mon_dict = {"nom": "Alice", "age": 25}
print(mon_dict.pop("age"))  # 25
print(mon_dict)  # {'nom': 'Alice'}
```

popitem() Supprime et renvoie la dernière paire clé-valeur

```
mon_dict = {"nom": "Alice", "age": 25}
print(mon_dict.popitem())  # ('age', 25)
print(mon_dict)  # {'nom': 'Alice'}
```

clear() Supprime toutes les paires clé-valeur d'un dictionnaire

```
mon_dict = {"nom": "Alice", "age": 25}
mon_dict.clear()
print(mon_dict)  # {}
```

setdefault(key, [default]) Renvoie la valeur associée à une clé. Si la clé n'existe pas, l'ajoute avec une valeur par défaut

```
mon_dict = {"nom": "Alice"}
print(mon_dict.setdefault("ville", "Paris"))  # Paris
print(mon_dict)  # {'nom': 'Alice', 'ville': 'Paris'}
```

Principales commandes pour modifier un ensemble (set)

Manipulation des éléments

add() Ajoute un élément à l'ensemble

```
mon_ensemble = {1, 2, 3}
mon_ensemble.add(4)
print(mon_ensemble)  # Résultat : {1, 2, 3, 4}
```

remove() Supprimer un élément

```
mon_ensemble = {1, 2, 3}
mon_ensemble.remove(2)
print(mon_ensemble)  # Résultat : {1, 3}
```

discard() Supprimer un élément

```
mon_ensemble = {1, 2, 3}
mon_ensemble.discard(2)
print(mon_ensemble)  # Résultat : {1, 3}

# Pas d'erreur si l'élément n'existe pas
mon_ensemble.discard(5)
print(mon_ensemble)  # Résultat : {1, 3}
```

pop() Supprimer et récupérer un élément aléatoire

```
mon_ensemble = {1, 2, 3, 4}
element = mon_ensemble.pop()
print(f"Élément supprimé : {element}")
print(mon_ensemble)  # L'ensemble sans l'élément supprimé
```

clear() Vider complètement un ensemble

```
mon_ensemble = {1, 2, 3}
mon_ensemble.clear()
print(mon_ensemble)  # Résultat : set()
```

Opérations sur les ensembles

union() Renvoie un ensemble contenant tous les éléments uniques des deux ensembles

```
ensemble1 = {1, 2, 3}
ensemble2 = {3, 4, 5}
union = ensemble1.union(ensemble2)
print(union)  # Résultat : {1, 2, 3, 4, 5}
```

intersection() Renvoie un ensemble contenant les éléments communs aux deux ensembles

```
ensemble1 = {1, 2, 3}
ensemble2 = {3, 4, 5}
intersection = ensemble1.intersection(ensemble2)
print(intersection)  # Résultat : {3}
```

difference() Renvoie un ensemble contenant les éléments présents dans le premier ensemble mais pas dans le second

```
ensemble1 = {1, 2, 3}
ensemble2 = {3, 4, 5}
difference = ensemble1.difference(ensemble2)
print(difference)  # Résultat : {1, 2}
```

symmetric_difference() Renvoie un ensemble contenant les éléments présents dans l'un ou l'autre des ensembles, mais pas les deux

```
ensemble1 = {1, 2, 3}
ensemble2 = {3, 4, 5}
symmetric_diff = ensemble1.symmetric_difference(ensemble2)
print(symmetric_diff)  # Résultat : {1, 2, 4, 5}
```

issubset() Vérifie si tous les éléments d'un ensemble sont dans un autre

```
ensemble1 = {1, 2}
ensemble2 = {1, 2, 3, 4}
print(ensemble1.issubset(ensemble2))  # Résultat : True
```

issuperset() Vérifie si tous les éléments d'un autre ensemble sont dans l'ensemble courant

```
ensemble1 = {1, 2, 3, 4}
ensemble2 = {1, 2}
print(ensemble1.issuperset(ensemble2))  # Résultat : True
```

Principales commandes pour modifier une chaînes de caractères (str)

strip() Supprime les espaces au début et à la fin d'une chaîne

```
texte = "  Bonjour  "
print(texte.strip())  # 'Bonjour'
```

lower() / upper() Convertit une chaîne en minuscules ou en majuscules

```
print("Bonjour".lower())  # 'bonjour'
print("Bonjour".upper())  # 'BONJOUR'
```

replace(old, new) Remplace une sous-chaîne par une autre

```
texte = "Bonjour tout le monde"
print(texte.replace("monde", "Python"))  # 'Bonjour tout le Python'
```

split(separator) Divise une chaîne en une liste selon un séparateur

```
texte = "Bonjour tout le monde"
print(texte.split(" "))  # ['Bonjour', 'tout', 'le', 'monde']
```

join(iterable) Concatène les éléments d'un itérable en une chaîne avec un séparateur

```
mots = ["Python", "est", "génial"]
print(" ".join(mots))  # 'Python est génial'
```

find(sub) Renvoie l'indice de la première occurrence d'une sous-chaîne

```
texte = "Python est génial"
print(texte.find("est"))  # 7
```

count(sub) Compte le nombre d'occurrences d'une sous-chaîne dans une chaîne

```
texte = "Python Python"
print(texte.count("Python"))  # 2
```

startswith(prefix) Vérifie si une chaîne commence par un préfixe

```
texte = "Python est génial"
print(texte.startswith("Python"))  # True
```

endswith(suffix) Vérifie si une chaîne se termine par un suffixe

```
texte = "Python est génial"
print(texte.endswith("génial"))  # True
```

Principales commandes pour modifier les tuples (tuple)

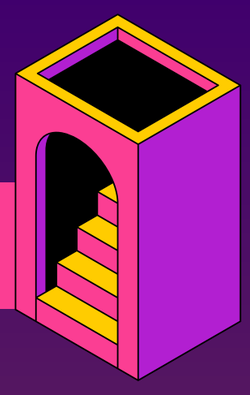
count(x) Compte le nombre de fois où une valeur apparaît dans un tuple

```
mon_tuple = (1, 2, 3, 2, 4)
print(mon_tuple.count(2))  # 2
```

index(x, [start], [end]) Renvoie l'indice de la première occurrence d'une valeur

```
mon_tuple = (1, 2, 3, 2, 4)
print(mon_tuple.index(2))  # 1
```

Python



match / case

Permet de créer des structures de contrôle conditionnelles

- **match** → déclare la variable ou l'expression à comparer
- **case** → définit les différents scénarios ou motifs de comparaison

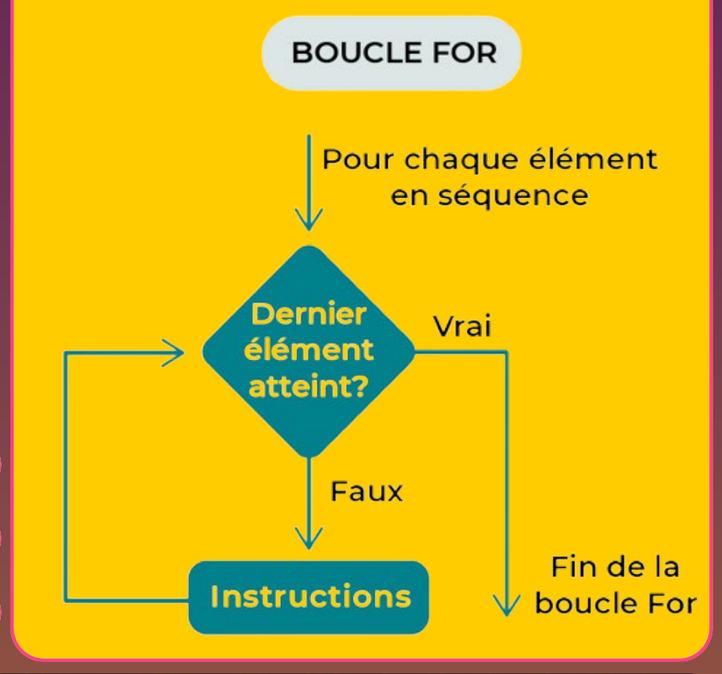
```python
note = 10

match note:
    case 18 | 19 | 20:
        print("Excellent")
    case 14 | 15 | 16 | 17:
        print("Très bien")
    case 10 | 11 | 12 | 13:
        print("Passable")
    case _:
        print("Échec")  # Passable
```

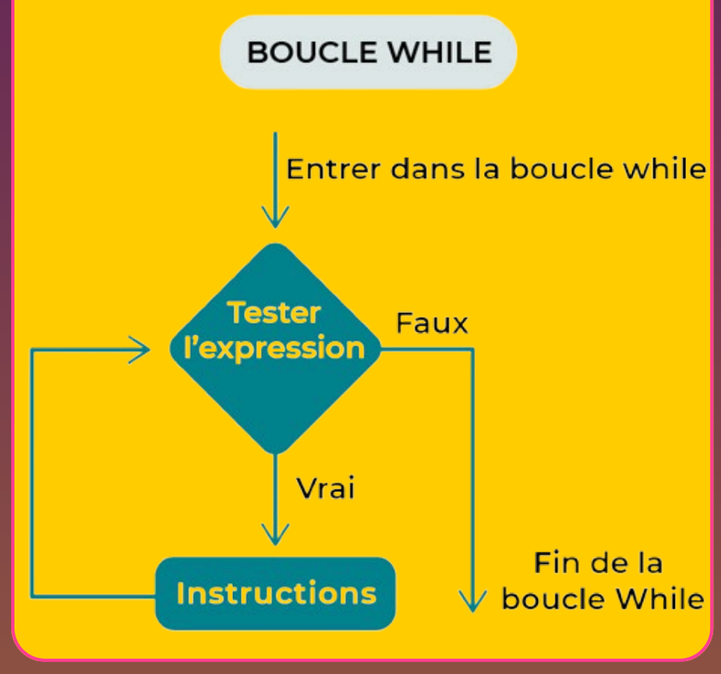
Les boucles

La boucle FOR

La boucle for en Python est utilisée pour itérer sur une séquence. À chaque itération, un élément de la séquence est traité

- **Range** → range(start, stop, step)
  - **start (optionnel)** → Point de départ de la séquence (inclus) Par défaut, c'est 0
  - **stop (obligatoire)** → Point d'arrêt de la séquence (exclu)
  - **step (optionnel)** → Incrément entre les valeurs Par défaut, c'est 1





La boucle WHILE

La boucle while est utilisée pour exécuter un bloc de code tant qu'une condition est vraie. Elle est particulièrement utile lorsqu'on ne sait pas à l'avance combien d'itérations sont nécessaires

```python
for i in range(1, 6):  # De 1 à 5 inclus
    print(i)
# Résultat :
# 1
# 2
# 3
# 4
# 5
```

```python
i = 1
while i <= 10:
    if i == 5:  # Si i atteint 5
        break  # Quitter la boucle
    print(i)
    i += 1
# Résultat :
# 1
# 2
# 3
# 4
```

- **break** → Interrompt la boucle immédiatement
- **continue** → Saute directement à l'itération suivante

```python
i = 0
while i < 5:
    i += 1
    if i == 3:  # Si i vaut 3
        continue  # Passer à l'itération suivante
    print(i)
# Résultat :
# 1
# 2
# 4
# 5
```

Les fonctions

Bloc de code réutilisable qui effectue une tâche spécifique. Elle est définie avec le mot-clé def et peut prendre des paramètres en entrée et retourner un résultat en sortie

```python
# Début du programme
if __name__ == "__main__":
    # Définir une fonction
    def saluer(nom):
        return f"Bonjour, {nom}!"

    # Appeler la fonction
    resultat = saluer("Alice")
    print(resultat)  # Résultat : Bonjour, Alice!
```

- **def** → Déclarer une fonction
- **saluer** → Nom de la fonction
- **(nom)** → Variables passées à la fonction
- **return** → Renvoie une valeur

try / except et else / finally

- **try** → Si une erreur survient dans ce bloc, l'exécution est immédiatement transférée au bloc except
- **except** → Bloc utilisé pour gérer les exceptions levées dans le bloc try
- **else** → Exécuté si aucune exception n'est levée
- **finally** → Exécuté dans tous les cas (exception ou non)

```python
try:
    nombre = int(input("Entrez un nombre : "))
    resultat = 10 / nombre
except ZeroDivisionError:
    print("Erreur : Division par zéro impossible.")
else:
    print(f"Résultat : {resultat}")
finally:
    print("Fin du programme.")
# Interaction :
# Entrez un nombre : 2
# Résultat : 5.0
# Fin du programme.
```

Import

Utilisé pour inclure des modules ou des bibliothèques externes dans votre programme

pip

Pip est le gestionnaire de packages Python. Il permet d'installer, de mettre à jour et de gérer des bibliothèques Python tierces

DBERARD