Python pour les débutants

Variables et chaînes de caractères

Les variables sont utilisées pour attribuer des étiquettes aux valeurs. Une chaîne est une série de caractères, entourée de guillemets simples ou doubles. Les f-strings de Python permettent d'utiliser des variables à l'intérieur de chaînes de caractères pour construire des messages dynamiques.

Bonjour le monde

```
print("Hello world !")
```

Bonjour le monde avec une variable

```
msg = "Hello world !" print(msg)
```

f-chaînes (utilisation de variables dans des chaînes)

```
prénom = "albert" nom =
"einstein
full_name = f"{first_name} {last_name}"
print(full_name)
```

Listes

Une liste stocke une série d'éléments dans un ordre particulier. Vous accédez aux éléments à l'aide d'un index ou à l'intérieur d'une boucle.

Faire une liste

```
vélos = ['trek', 'redline', 'giant']
```

Obtenir le premier élément d'une liste

```
first_bike = bikes[0]
```

Obtenir le dernier élément d'une liste

```
last_bike = bikes[-1]
```

Boucler une liste

```
pour vélo dans vélos :
    print(vélo)
```

Ajouter des éléments à une liste

```
bikes = []
bikes.append('trek')
bikes.append('redline')
bikes.append('giant')
```

Établir des listes numériques

```
carrés = []
```

Découpage d'une liste

```
finishers = ['sam', 'bob', 'ada', 'bea'] first_two
= finishers[:2]
```

Copier une liste

```
copy_of_bikes = bikes[ :]
```

Tuples

Les tuples sont similaires aux listes, mais les éléments d'un tuple ne peuvent pas être modifiés.

Création d'un tuple

```
dimensions = (1920, 1080)
résolutions = ('720p', '1080p', '4K')
```

Si les déclarations

Les instructions "si" sont utilisées pour tester des conditions particulières et réagir de manière appropriée.

Tests conditionnels

égal	x == 42
n'est pas égal	x != 42
supérieur à	x > 42
ou égal à	x >= 42
inférieur à	x < 42
ou égal à	x <= 42

Tests conditionnels avec des listes

```
trek" dans vélos
"surly" pas dans
vélos
```

Attribution de valeurs booléennes

```
game_active = True
can_edit = False
```

Un simple test if

```
si âge >= 18 :
    print("Vous pouvez voter !")
```

Déclarations "si", "if" et "else" (si, si, si, si)

```
si âge < 4 :
    ticket_price = 0
elif age < 18 :
    ticket_price = 10
elif age < 65 :
    ticket_price = 40
else :
    prix_du_billet = 15
```

Un dictionnaire simple

```
alien = {'color' : 'green', 'points' : 5}
```

Accès à une valeur

```
print(f "La couleur de l'extraterrestre est
{alien['couleur']}.")
```

Ajout d'une nouvelle paire clé-valeur

```
alien['x_position'] = 0
```

Passer en revue toutes les paires clé-valeur

```
fav_numbers = {'eric' : 7, 'ever' : 4, 'erin' : 47}

for name, number in fav_numbers.items() :
    print(f"{nom} aime {numéro}.")
```

Passer en revue toutes les touches

```
fav_numbers = {'eric' : 7, 'ever' : 4, 'erin' : 47}

for name in fav_numbers.keys() :
    print(f"{nom} aime un nombre.")
```

Passer en revue toutes les valeurs

```
fav_numbers = {'eric' : 7, 'ever' : 4, 'erin' : 47}

for number in fav_numbers.values() :
    print(f"{number} is a favorite.")
```

Données de l'utilisateur

Vos programmes peuvent demander à l'utilisateur d'entrer des données. Toutes les entrées sont stockées sous la forme d'une chaîne de caractères.

Demande d'une valeur

```
name = input("Quel est votre nom ? ") print(f
"Bonjour, {nom} !")
```

Invitation à la saisie numérique

```
age = input("Quel âge a v e z - v o u s
? ") age = int(age)

pi = input("Quelle est la valeur de pi ? ")
pi = float(pi)
```

Cours accéléré sur Python

Une introduction pratique à la programmation, basée sur des projets

ehmatthes.github.io/pcc_3e

Boucles "While"

Une boucle while *répète un bloc de code tant qu'une certaine condition est remplie. Les boucles while sont particulièrement utiles lorsque vous ne pouvez pas savoir à l'avance combien de fois une boucle doit être exécutée.*

Une simple boucle while

```
valeur_actuelle = 1
while valeur_actuelle <= 5 :
    print(valeur_actuelle)
    valeur_actuelle += 1
```

Permettre à l'utilisateur de choisir le moment de quitter le système

```
msg = ''
while msg != 'quit' :
    msg = input("Quel est votre message ? ")

    if msg != 'quit' :
        print(msg)
```

Fonctions

Les fonctions sont des blocs de code nommés, conçus pour effectuer une tâche spécifique. L'information transmise à une fonction est appelée argument, et l'information reçue par une fonction est appelée paramètre.

Une fonction simple

```
def greet_user() :
    """Afficher un message d'accueil
    simple."""  print("Hello !")

greet_user()
```

Passage d'un argument

```
def greet_user(nom d'utilisateur) :
    """Afficher un message d'accueil
    personnalisé."""  print(f "Hello, {username} !")

greet_user('jesse')
```

Valeurs par défaut des paramètres

```
def make_pizza(topping='pineapple') :
    """Faire une pizza à garniture unique.""
    print(f "Avoir une p i z z a  {à garniture} !")

make_pizza()
make_pizza('champignon')
```

Renvoi d'une valeur

```
def add_numbers(x, y) :
    """Additionner deux nombres et renvoyer la
    somme."""  return x + y

sum = add_numbers(3, 5)
print(sum)
```

Classes

Une classe définit le comportement d'un objet et le type d'informations qu'il peut stocker. Les informations d'une classe sont stockées dans des attributs, et les fonctions qui appartiennent à une classe sont appelées méthodes. Une classe enfant hérite des attributs et des méthodes de sa classe mère.

Créer une classe de chiens

```
classe Chien :
    """Représenter un chien.""

    def init (self, name) : """Initialise
        l'objet chien."" self.name =
        name

    def sit(self) :
        """Simuler la position assise.""
        print(f"{self.name} est assis.")

mon_chien = Chien('Peso')

print(f"{mon_chien.nom} est un super chien !")
mon_chien.assis()
```

Héritage

```
classe SARDog(Chien) :
    """Représenter un chien de recherche.""

    def init (self, name) : """Initialise
        le sardog."" super(). init
        (name)

    def search(self) :
        """Simuler la recherche."""
        print(f"{self.name} est en train de
        chercher.")

mon_chien = SARDog('Willie')

print(f"{mon_chien.nom} est un chien de
recherche") mon_chien.assis()
mon_chien.recherche()
```

Compétences infinies

Si vous aviez des compétences infinies en programmation, que construiriez-vous ?

Lorsque vous apprenez à programmer, il est utile de réfléchir aux projets concrets que vous aimeriez créer. C'est une bonne habitude de garder un carnet d'"idées" auquel vous pouvez vous référer chaque fois que vous voulez commencer un nouveau projet.

Si vous ne l'avez pas encore fait, prenez quelques minutes pour décrire trois projets que vous aimeriez créer. Au cours de votre apprentissage, vous pouvez écrire de petits programmes en rapport avec ces

Travailler avec des fichiers

Vos programmes peuvent lire des fichiers et écrire dans des fichiers.
La bibliothèque pathlib facilite le travail avec les fichiers et les répertoires. Une fois le chemin défini, vous pouvez utiliser les méthodes read_text() *et* write_text().

Lire le contenu d'un fichier

La méthode read_text() *lit l'intégralité du contenu d'un fichier. Vous pouvez ensuite diviser le texte en une liste de lignes individuelles, puis traiter chaque ligne comme vous le souhaitez.*

```
from pathlib import Path

path = Path('siddhartha.txt')
contents = path.read_text() lines
= contents.splitlines()

pour ligne dans lignes :
    print(ligne)
```

Écriture dans un fichier

```
path = Path('journal.txt')

msg = "J'aime la programmation")
path.write_text(msg)
```

Exceptions

Les exceptions vous aident à réagir de manière appropriée aux erreurs susceptibles de se produire. Le code susceptible de provoquer une erreur est placé dans le bloc try. Le code qui doit être exécuté en réponse à une erreur est placé dans le bloc except. Le code qui ne doit être exécuté que si le bloc try a réussi est placé dans le bloc else.

Rattraper une exception

```
prompt = "Combien de billets avez-vous besoin ?
"num_tickets = input(prompt)

essayer :
    num_tickets = int(num_tickets)
except ValueError :
    print("Veuillez réessayer.")
else :
    print("Vos billets sont en cours
    d'impression")
```

Le zen de Python

La simplicité vaut mieux que la complexité

Si vous avez le choix entre une solution simple et une solution complexe, et que les deux fonctionnent, choisissez la solution simple. Votre code sera plus facile à maintenir, et il sera plus facile pour vous et pour les autres de construire sur ce code par la suite.

Aide-mémoire Python pour les débutants - Listes

Qu'est-ce qu'une liste ?

Une liste stocke une série d'éléments dans un ordre particulier. Les listes vous permettent de stocker des ensembles d'informations en un seul endroit, qu'il s'agisse de quelques éléments ou de millions d'éléments. Les listes sont l'une des fonctionnalités les plus puissantes de Python, facilement accessible aux nouveaux programmeurs, et elles relient de nombreux concepts importants de la programmation.

Définition d'une liste

Utilisez des crochets pour définir une liste et des virgules pour séparer les différents éléments de la liste. Utilisez des noms pluriels pour les listes, afin d'indiquer clairement que la variable représente plus d'un élément.

Faire une liste

```
users = ['val', 'bob', 'mia', 'ron', ' ned']
```

Accès aux éléments

Les éléments individuels d'une liste sont accessibles en fonction de leur position, appelée indice. L'indice du premier élément est 0, l'indice du deuxième élément est 1, et ainsi de suite. Les indices négatifs font référence aux éléments situés à la fin de la liste. Pour obtenir un élément particulier, écrivez le nom de la liste, puis l'indice de l'élément entre crochets.

Obtenir le premier élément

```
premier_utilisateur = utilisateurs[0]
```

Obtention du deuxième élément

```
deuxième_utilisateur = utilisateurs[1]
```

Récupérer le dernier élément

```
newest_user = users[-1]
```

Modifier des éléments individuels

Une fois que vous avez défini une liste, vous pouvez modifier la valeur de chacun de ses éléments. Pour ce faire, vous devez vous référer à l'index de l'élément que vous souhaitez modifier.

Modification d'un élément

```
users[0] = 'valerie'
```

Ajout d'éléments

Vous pouvez ajouter des éléments à la fin d'une liste ou les insérer où vous le souhaitez dans une liste. Cela vous permet de modifier des listes existantes ou de commencer avec une liste vide et d'y ajouter des éléments au fur et à mesure que le programme se développe.

Ajout d'un élément à la fin de la liste

```
users.append('amy')
```

Commencer par une liste vide

```
users = []
users.append('amy')
users.append('val')
users.append('bob')
users.append('mia')
```

Insérer des éléments à une position particulière

```
users.insert(0, 'joe')
users.insert(3, 'bea')
```

Suppression d'éléments

Vous pouvez supprimer des éléments en fonction de leur position dans une liste ou de leur valeur. Si vous supprimez un élément par sa valeur, Python ne supprime que le premier élément qui a cette valeur.

Suppression d'un élément en fonction de sa position

```
del users[-1]
```

Retrait d'un élément en fonction de sa valeur

```
users.remove('mia')
```

Éléments d'éclatement

Si vous souhaitez travailler avec un élément que vous retirez de la liste, vous pouvez "pop" l'élément. Si vous considérez la liste comme une pile d'éléments, pop() *retire un élément du sommet de la pile.*

Par défaut, pop() *renvoie le dernier élément de la liste, mais vous pouvez également extraire des éléments à partir de n'importe quelle position dans la liste.*

Extraire le dernier élément d'une liste

```
most_recent_user = users.pop()
print(most_recent_user)
```

Extraire le premier élément d'une liste

```
premier_utilisateur =
users.pop(0)
print(premier_utilisateur)
num_users = len(users)
print(f "Nous avons {num_users} utilisateurs.")
```

Trier une liste

La méthode sort() *modifie l'ordre d'une liste de façon permanente. La fonction* sorted() *renvoie une copie de la liste, en laissant la liste originale inchangée.*

Vous pouvez trier les éléments d'une liste par ordre alphabétique ou par ordre alphabétique inverse. Vous pouvez également inverser l'ordre initial de la liste. N'oubliez pas que les lettres minuscules et majuscules peuvent influer sur l'ordre de tri.

Trier une liste de façon permanente

```
utilisateurs.trier()
```

Tri permanent d'une liste par ordre alphabétique inversé

```
users.sort(reverse=True)
```

Tri temporaire d'une liste

```
print(sorted(users))
print(sorted(users, reverse=True))
```

Inverser l'ordre d'une liste

```
utilisateurs.reverse()
```

Boucler une liste

Les listes peuvent contenir des millions d'éléments, c'est pourquoi Python propose un moyen efficace de parcourir en boucle tous les éléments d'une liste. Lorsque vous mettez en place une boucle, Python extrait chaque élément de la liste un par un et l'affecte à une variable temporaire, à laquelle vous donnez un nom. Ce nom doit être la version singulière du nom de la liste.

Le bloc de code indenté constitue le corps de la boucle, où vous pouvez travailler sur chaque élément individuel. Toutes les lignes non indentées sont exécutées après la fin de la boucle.

Impression de tous les éléments d'une liste

```
for user in users :
    print(user)
```

Impression d'un message pour chaque élément, et d'un message séparé après chaque élément

```
for user in users :
    print(f"\nBienvenue,
    {utilisateur} !")
    print("Nous sommes ravis que vous nous ayez
    rejoints !")
print("\nBienvenue, nous sommes heureux de vous
voir tous !")
```

La fonction range()

Vous pouvez utiliser la fonction `range()` *pour travailler efficacement avec un ensemble de nombres. La fonction* `range()` *commence à 0 par défaut et s'arrête un nombre en dessous du nombre qui lui a été transmis. Vous pouvez utiliser la fonction* `list()` *pour générer efficacement une grande liste de nombres.*

Impression des nombres 0 à 1000

```
for number in range(1001) :
    print(number)
```

Impression des chiffres de 1 à 1000

```
for number in range(1, 1001) :
    print(number)
```

Faire une liste de nombres de 1 à 1 million

```
numbers = list(range(1, 1_000_001))
```

Statistiques simples

Il existe un certain nombre d'opérations statistiques simples que vous pouvez effectuer sur une liste contenant des données numériques.

Trouver la valeur minimale dans une liste

```
âges = [93, 99, 66, 17, 85, 1, 35, 82, 2, 77]
le plus jeune = min(ages)
```

Recherche de la valeur maximale

```
âges = [93, 99, 66, 17, 85, 1, 35, 82, 2, 77]
oldest = max(ages)
```

Trouver la somme de toutes les valeurs

```
âges = [93, 99, 66, 17, 85, 1, 35, 82, 2, 77]
total_years = sum(ages)
```

Découpage d'une liste

Vous pouvez travailler avec n'importe quel sous-ensemble d'éléments d'une liste. Une partie d'une liste est appelée "tranche". Pour découper une liste, commencez par l'indice du premier élément souhaité, puis ajoutez deux points et l'indice du dernier élément souhaité. Omettez le premier indice pour commencer au début de la liste, et omettez le deuxième indice pour découper la liste jusqu'à la fin.

Obtenir les trois premiers éléments

```
finishers = ['kai', 'abe', 'ada', 'gus', 'zoe']
first_three = finishers[:3]
```

Obtenir les trois éléments du milieu

```
middle_three = finishers[1:4]
```

Obtenir les trois derniers éléments

```
last_three = finishers[-3 :]
```

Copier une liste

Pour copier une liste, il faut faire une tranche qui commence au premier élément et se termine au dernier. Si vous essayez de copier une liste sans utiliser cette approche, tout ce que vous ferez à la liste copiée affectera également la liste d'origine.

Faire une copie d'une liste

```
finishers = ['kai', 'abe', 'ada', 'gus', 'zoe']
copy_of_finishers = finishers[ :]
```

Compréhension de listes

Vous pouvez utiliser une boucle pour générer une liste basée sur une plage de nombres ou sur une autre liste. Il s'agit d'une opération courante, c'est pourquoi Python offre un moyen plus efficace de la réaliser. Les compréhensions de listes peuvent sembler compliquées au premier abord ; si c'est le cas, utilisez l'approche de la boucle `for` *jusqu'à ce que vous soyez prêt à utiliser les compréhensions.*

Pour écrire une compréhension, définissez une expression pour les valeurs que vous souhaitez stocker dans la liste. Ensuite, écrivez une boucle `for` *pour générer les valeurs d'entrée nécessaires à la constitution de la liste.*

Utilisation d'une boucle pour générer une liste de nombres carrés

```
carrés = []
for x in range(1, 11) : square
    = x**2
    squares.append(square)
```

Utiliser une compréhension pour générer une liste de nombres carrés

```
carrés = [x**2 for x in range(1, 11)]
```

Utilisation d'une boucle pour convertir une liste de noms en majuscules

```
names = ['kai', 'abe', 'ada', 'gus', 'zoe']

upper_names = [] for
name in names :
    noms_supérieurs.append(nom.supérieur())
```

Utilisation d'une compréhension pour convertir une liste de noms en majuscules

```
names = ['kai', 'abe', 'ada', 'gus', 'zoe']

noms_supérieurs = [nom.supérieur() pour nom dans
noms]
```

Styliser votre code

La lisibilité compte

Respectez les conventions de mise en forme de Python :

Tuples

Un tuple est semblable à une liste, sauf que vous ne pouvez pas modifier les valeurs d'un tuple une fois qu'il est défini. Les n-uplets permettent de stocker les informations qui ne doivent pas être modifiées pendant la durée de vie d'un programme. Les tuples sont généralement désignés par des parenthèses.

Vous pouvez écraser un tuple entier, mais vous ne pouvez pas modifier les valeurs des éléments individuels.

Définition d'un tuple

```
dimensions = (800, 600)
```

Boucler un tuple

```
pour dimension dans dimensions :
    print(dimension)
```

Écraser un tuple

```
dimensions = (800, 600)
print(dimensions)

dimensions = (1200, 900)
print(dimensions)
```

Visualisation du code

Lorsque vous vous familiarisez avec des structures de données telles que les listes, il est utile de visualiser la manière dont Python traite les informations dans votre programme. Le Tuteur Python est un excellent outil pour voir comment Python suit les informations contenues dans une liste. Essayez d'exécuter le code suivant sur pythontutor.com, puis exécutez votre propre code.

Construire une liste et imprimer les éléments de la liste

```
chiens = []
chiens.append('willie')
chiens.append('hootz')
chiens.append('peso')
chiens.append('goblin')

for dog in dogs : print(f
    "Bonjour {chien} !")
print("J'aime ces chiens !")

print("Voici mes deux premiers chiens :")
old_dogs = dogs[:2]
for old_dog in old_dogs :
    print(old_dog)

del dogs[0]
dogs.remove('peso')
print(dogs)
```