

Introduction python

cookbook pour Ini03

Julien Ithurbide

21.11.2022

Table des matières

Préface	2
Structogramme	2
Départ / Fin	2
Instructions	2
les entrées/sorties	2
Les conditions	3
Les bases	3
Compter avec python	3
Les chaînes de caractères	4
Tout est objet	4
Les variable	5
L'accès à une partie de chaîne	5
Les entrée/sorties	6
L'instruction print()	6
L'instruction input()	6
Les boucles	7
True, False, None	7
La boucle for	8
La boucle while	9
Les conditions	10
Si	10
Si sinon	11
Les fonctions	11
Les classes	12

Préface

Ce livre est destiné aux élèves de première année CFC en informatique. Il a été créé afin d'avoir une référence sur le langage python utilisé au sein du CPNV.

Ce livre n'est en aucun cas une référence pour python. Le langage est bien plus complexe et avancé que ce que le livre présente. Il permettra cependant, d'avoir une bonne connaissance de la syntaxe, et des fonctions de base du langage.

Structogramme

Un des moyen de présenter un programme (ou algorythme) et de le représenter sous forme d'un schéma. Nous allons voir le structogramme qui est le plus efficace pour présenter le déroulement d'un programme

Départ / Fin

Le diagramme comme et fin toujours avec ce nœud.

Départ / Fin

Instructions

Chaque instruction (ligne de code avec calcule ou autre) sera représenté comme tel :

une instruction

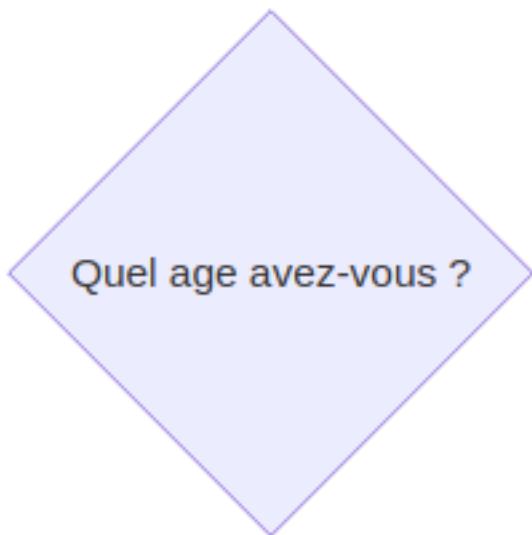
les entrées/sorties

Lorsque l'on veut écrire du texte à l'écran, ou demander quelque chose à l'utilisateur, nous utilisons ceci :

Merci de donner un nombre

Les conditions

Lorsque que l'on doit poser une question, nous utilisons le losange comme tel :



Les bases

Compter avec python

L'interpréteur python permet de faire des calculs de manière intuitif. L'addition, la soustraction, la multiplication sont simple. La division est un peu particulière. Regardons ceci de plus prêt. Voici un tableau qui représente les opérations possibles.

Description	Notation
Addition	+
Soustraction	-
Multiplication	*
Division	/
Exponentiel	**
Division entière	//
Reste de la division entière	%

```
1 >>> 2 + 3
2 5
3
4 >>> 5 - 7
5 -2
6
7 >>> 3 * 4
8 12
9
10 >>> 5 / 3
11 1.6666666666666667
12
13 >>> 2 ** 4
14 16
15
16 >>> 5 // 3
17 1
18
19 >>> 5 % 3
20 2
```

Les chaînes de caractères

Il est aussi possible de travailler avec une chaîne de caractères. Pour cela, il suffit de placer le texte voulu entre simple ou double guillemet.

```
1 >>> "une simple chaine"
2 'une simple chaine'
```

Tout est objet

En python, une chaine de caractère, un nombre, un tableau enfin tout est considéré comme un objet. Il n'y a pas besoin de spécifier le type. Python détecte le contenu et donne un type de manière automatique.

Une instruction permet de connaître le type d'un objet. Cette instruction est `type()`

```
1 >>> type(8)
2 <class 'int'>
3 >>> type("ma machine")
4 <class 'str'>
5 >>>
```

Les variable

Une variable est un nom associé à un emplacement de la mémoire. C'est comme une boîte que l'on identifie par une étiquette.

La instruction `a = 3` signifie que j'ai une variable « a » associée à la valeur 3.

Voici un premier exemple :

```
1 >>> a = 3
2 >>> b = 5
3 >>> c = a + b
4 >>> type(a)
5 <class 'int'>
6 >>> type(b)
7 <class 'int'>
8 >>> type(c)
9 <class 'int'>
10 >>> c
11 8
```

Les variables sont très pratique pour stocker des valeurs et utiliser leur nom plus tard dans le code. Il existe plusieurs types de variables. Les nombres entiers vu par python comme la classe `int`. Les nombre à virgule vu par python comme la classe `float`. Les chaînes de caractères vu par python comme `str` raccourcis du mot string. Les booléens vu par python comme la classe `bool`.

L'accès à une partie de chaîne

L'opérateur `[:]` permet de découper une chaîne de caractère. La partie avant les “:” permet de définir le début de la nouvelle chaine. La partie après les “:” défini la fin.

Voici un exemple, commençons par créer une variable avec comme contenu une chaîne de caractère.

```
1 >>> ma_chaine = "une simple chaine"
2 >>> ma_chaine
3 'une simple chaine'
```

Si nous voulons seulement la chaîne “une simple”, nous pouvons extraite celle-ci avec le code suivant :

```
1 >>> ma_nouvelle_chaine = ma_chaine[:-7]
2 >>> ma_nouvelle_chaine
3 'une simple'
```

Si nous voulons seulement la chaîne “simple chaîne”, le code suivant sera utilisé :

```
1 >>> ma_deuxime_chaine = ma_chaine[4:]
```

```
2 >>> ma_deuxime_chaine  
3 'simple chaine'
```

Comme vous le remarquez, si nous laissons la partie du début vide, la nouvelle chaîne commence au début. Si nous laissons la deuxième partie vide, nous allons jusqu'à la fin.

Il est aussi possible d'avoir accès à un caractère spécifique tel que :

```
1 >>> ma_chaine  
2 'une simple chaine'  
3 >>> ma_chaine[0]  
4 'u'  
5 >>> ma_chaine[1]  
6 'n'  
7 >>> ma_chaine[2]  
8 'e'
```

Les entrée/sorties

Afin de communiquer avec l'utilisateur de votre programme, il va falloir utiliser deux fonctions de base.

L'instruction `print()`

L'instruction `print()` permet d'écrire quelque chose à l'écran. C'est une fonction qui prend en paramètre le texte que l'on veut afficher.

Exemple :

```
1 print("bonjour le monde")
```

Il est possible d'utiliser l'instruction `print` pour afficher la valeur d'une variable.

```
1 print("La somme vaut", a+b) # Affiche la somme  
2 print("Le produit vaut", a*b) # Affiche le produit
```

Il est possible d'utiliser directement le nom des variables dans le texte comme ceci :

```
1 print(f"La somme vaut {a+b}") # Affiche la somme
```

L'instruction `input()`

Il est possible de demander à l'utilisateur une valeur qu'il devra entrer au clavier. L'instruction permettant ceci s'appelle `input()`. `input()` met le programme en pause et attend de l'utilisateur

un message tapé au clavier (qu'il termine en appuyant sur la touche « Entrée »).

Attention, le message est une chaîne de caractères. Pour transformer ce que l'utilisateur a entré au clavier, il faut utiliser une des instructions suivantes : **int()** ou **float()** en fonction du besoin.

Voici un petit programme qui demande le prénom et l'âge de l'utilisateur et affiche un message du style « Bonjour Kevin » puis « Tu es mineur/majeur » selon l'âge.

```
1 prenom = input("Comment t'appelles-tu ? ")
2 print(f"Bonjour {prenom}")
3 age_chaine = input("Quel age as-tu ? ")
4 age = int(age_chaine)
5 if age >= 18:
6     print("Tu es majeur !")
7 else:
8     print("Tu es mineur !")
```

Un autre exemple qui permet de demander un nombre décimal.

```
1 pourcent = input("Donnez moi un pourcentage : ")
2 # ici, on transforme en float
3 try:
4     pourcent_float = float(pourcent)
5 except:
6     print("An exception occurred")
7     pourcent_float = -1
8 while pourcent_float == -1:
9     try:
10         pourcent = input("Donnez moi un pourcentage : ")
11         pourcent_float = float(pourcent)
12     except:
13         pourcent_float = -1
```

Les boucles

True, False, None

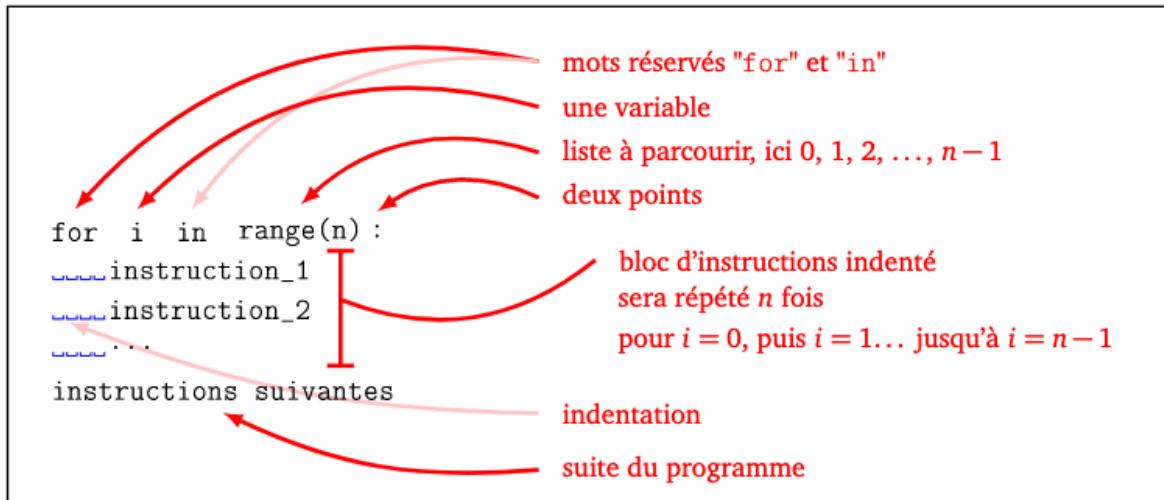
Avant de commencer les boucles, il nous faut parler de trois type de variable particulière.

Une variable de type **None** est une variable non encore utilisée. Elle ne représente rien.

Une variable de type booléen permet de stocker le résultat d'une opération logique. Il y a deux valeurs possible **True** (vrai) ou **False** (faux).

La boucle for

Lorsque l'on a besoin de répéter une opération plusieurs fois (le nombre de fois est connu), nous pouvons utiliser une boucle **for** qui s'écrit comme tel :



Si l'on veut répéter une action :

```

1 >>> for x in range(0,3):
2     ...     print(x)
3 ...
4 0
5 1
6 2
  
```

Si nous n'avons pas besoin de la variable temporaire, nous pouvons utiliser le caractère “_”.

```

1 >>> for _ in range(0,3):
2     ...     print("hello")
3 ...
4 hello
5 hello
6 hello
  
```

Cette boucle permet aussi de parcourir une chaîne de caractère.

```

1 >>> ma_chaine = "une simple chaîne"
2 >>> for c in ma_chaine:
3     ...     print(c)
4 ...
5 u
6 n
7 e
  
```

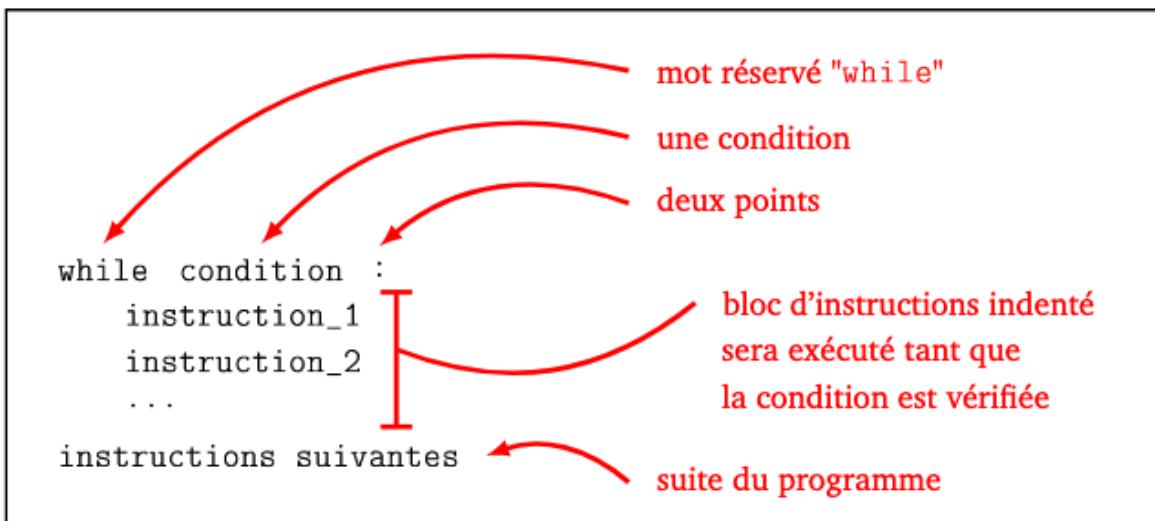
```

8
9 s
10 i
11 m
12 p
13 l
14 e
15
16 c
17 h
18 a
19 i
20 n
21 e

```

La boucle while

Il existe une autre boucle qui se répète tant qu'une condition est "vrai". Voici sa syntaxe :



La boucle **while** est souvent utilisée pour tester l'entrée d'un utilisateur. Tant que l'entrée faite ne correspond pas à ce que nous voulons on répète. Voici un exemple permettant de s'assurer que l'utilisateur a bien entré un entier :

```

1 >>> str_nombre_entier = input("Merci de donner un entier : ")
2 Merci de donner un entier : b
3 >>> if str_nombre_entier.isdigit():
4     ...     nombre_entier = int(str_nombre_entier)
5 ... else:
6     ...     nombre_entier = -1
7 ...

```

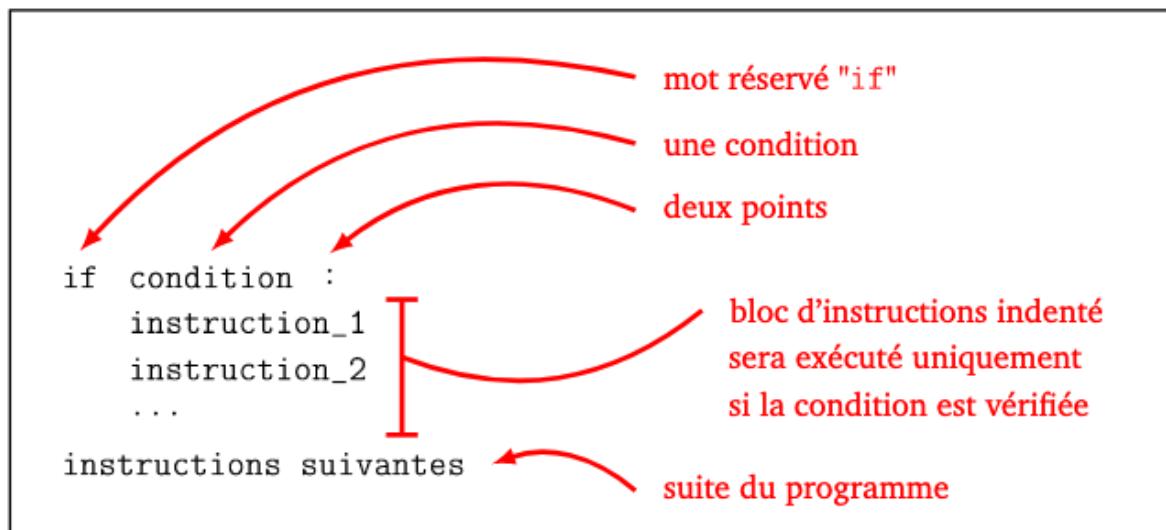
```

8  >>> while nombre_entier <= 0 or nombre_entier >= 100 :
9    ...     str_nombre_entier = input("Merci de bien vouloir enter un
10   ...       entier : ")
11  ...       if str_nombre_entier.isdigit():
12  ...           nombre_entier = int(str_nombre_entier)
13  ...       else:
14  ...           nombre_entier = -1
15 ...
16 Merci de bien vouloir enter un entier : c
16 Merci de bien vouloir enter un entier : asas
17 Merci de bien vouloir enter un entier : 3

```

Les conditions

Si



Si l'on recherche à tester deux valeurs, comme par exemple si la variable `a` est < que `b`, il nous faut utiliser l'instruction `if`.

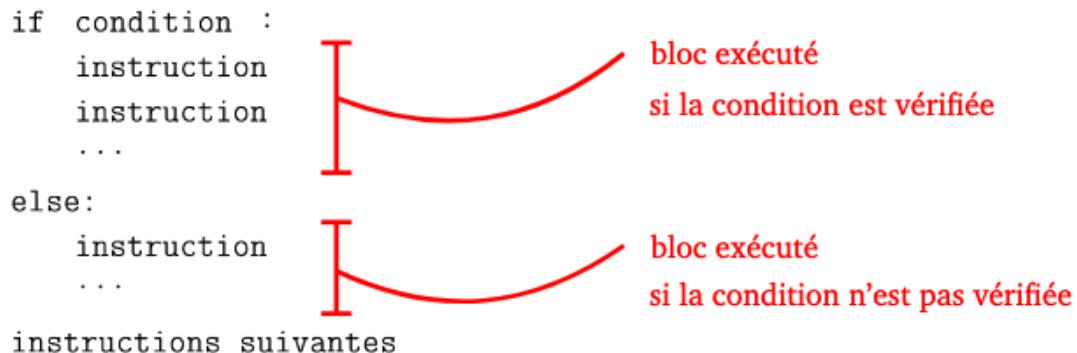
```

1  >>> a = 10
2  >>> b = 5
3  >>> if a < b :
4    ...     print("plus petit")
5  ...

```

Dans ce cas, rien ne se passe. Car `a` est plus grand que `b`.

Si sinon



Pour résoudre le problème, ci dessus, nous pouvons utiliser l'instruction **else** qui permet d'exécuter une instruction dans les autres cas. Pour cela, reprenons l'exemple précédent et ajoutons **else**.

```

1 >>> a = 10
2 >>> if a < b :
3 ...     print("plus petit")
4 ... else:
5 ...     print("plus grand")
6 ...
7 plus grand
  
```

Ici, le message **plus grand** s'affiche car **a** est bien plus grand que **b**.

Les fonctions

Lorsqu'un bout de code est souvent utilisé, il est possible de faire une fonction afin d'éviter :

- La redondance de code
- La propagation d'erreur

Une fonction désigne en programmation un « sous-programme » permettant d'effectuer des opérations répétitives. Au lieu d'écrire le code complet autant de fois que nécessaire, on crée une fonction que l'on appellera pour l'exécuter, ce qui peut aussi alléger le code, le rendre plus lisible.¹

¹Source wikipedia

Une fonction en python commence par le terme `def` suivit du nom que l'on veut donner à la fonction. Le code suit avec une tabulation tel que :

```
1 def dire_bonjour():
2     print("Bonjour!")
```

Pour appeler la fonction dans votre code, il suffit de taper la ligne : `dire_bonjour()`

Ce qui donne :

```
1 def dire_bonjour():
2     print("Bonjour!")
3
4 dire_bonjour()
```

Dans certains cas, nous avons besoin de passer des informations (ce qu'on appelle paramètres) à notre fonction. Voici un exemple de comme faire :

```
1 def additionner(a,b) :
2     print("La somme vaut :", a+b)
3
4 additionner(2,3)
```

Ici, l'exemple n'est pas complet. Le but d'une fonction est de pouvoir être utilisable plusieurs fois. Ainsi, faire un `print` dedant n'est pas correcte. De plus, une fonction doit faire une et une seule chose. Ici, nous affichons quelque chose à l'écran et faisons une addition. Ce n'est pas correcte. Voici donc une fonction qui permet de renvoyer le résultat à notre programme. Et c'est le programme qui s'occupe de l'affichage :

```
1 def additionner(a,b) ->int :
2     return a+b
3
4 resultat = additionner(2,3)
5 print("Le résultat vaut :", resultat)
```

Ici, il y a deux concepts supplémentaires. Le premier, le terme `return` qui permet de renvoyer à notre programme une valeur. Et le second, le terme `-> int`. Celui-ci permet d'indiquer que nous envoyons un entier en retour de notre fonction.

Les classes

En Python, les classes sont un concept de programmation objet qui permet de définir une structure de données et des méthodes associées pour manipuler ces données. Une classe est à peu près équivalente à une forme ou une convention pour créer des objets.

Les classes ont plusieurs utilisations importantes en Python :

1. **Modéliser les objets réels** : Les classes permettent de modéliser les objets du monde réel, comme des personnes, des véhicules, des animaux, etc. Vous pouvez définir des attributs (ou propriétés) pour ces objets, ainsi que des méthodes qui les manipulent ou les modifient.
2. **Grouper des données et des fonctions** : Les classes permettent de grouper des données et des fonctions associées pour un objet spécifique. Cela rend votre code plus facile à comprendre et à maintenir, car vous pouvez organiser vos données et vos méthodes de manière logique.
3. **Créer des instances** : Une classe peut être utilisée pour créer des instances (ou objets) qui partagent les mêmes attributs et méthodes que la classe elle-même. Cela permet de réutiliser du code et d'éviter la duplication.

Voici la syntaxe générale pour définir une classe en Python :

```

1 class NomDeLaClasse:
2     def __init__(self, attribut1, attribut2):
3         self.attribut1 = attribut1
4         self.attribut2 = attribut2
5     def methode(self):
6         # Code de la méthode

```

Les éléments clés à comprendre pour utiliser les classes en Python sont :

- **Constructeur (`__init__`)**: C'est la méthode qui est appelée lorsque vous créez une instance d'une classe. Elle permet de initialiser les attributs de l'objet.
- **Attributs** (ou propriétés) : Les attributs sont des variables définies dans la classe et qui peuvent être accessibles via les instances de la classe.
- **Méthodes** : Les méthodes sont des fonctions définies dans la classe qui peuvent manipuler les attributs de l'objet ou exécuter d'autres opérations.

Voici un exemple simple pour illustrer cela :

```

1 class Personne:
2     def __init__(self, nom, age):
3         self.nom = nom
4         self.age = age
5
6     def parler(self):
7         print(f"Bonjour, je m'appelle {self.nom} et j'ai {self.age} ans
8             .")
9
10 pierre = Personne("Pierre", 30)
11 pierre.parler() # Affichera : "Bonjour, je m'appelle Pierre et j'ai 30
12 ans."

```

En Python, l'héritage de classe (ou inheritance) permet à une classe d'étendre ou de modifier les comportements et les attributs d'une autre classe. Cela permet de créer une hiérarchie de classes qui partagent des caractéristiques communes.

Voici un exemple avec des animaux :

```

1  class Animal:
2      def __init__(self, name):
3          self.name = name
4
5      def sound(self):
6          pass # Méthode à implémenter dans les classes enfants
7
8  class Mammal(Animal):
9      def __init__(self, name, has_fur=True):
10         super().__init__(name)
11         self.has_fur = has_fur
12
13     def sound(self):
14         print(f"{self.name} fait un son de mammifère.")
15
16 class Dog(Mammal):
17     def __init__(self, name="Fido"):
18         super().__init__(name)
19
20     def sound(self):
21         print(f"{self.name} aboi.")
22
23 class Cat(Mammal):
24     def __init__(self, name="Whiskers", has_fur=False):
25         super().__init__(name, has_fur)
26
27     def sound(self):
28         print(f"{self.name} miaou.")

```

Dans cet exemple :

- La classe `Animal` est la classe parente (ou supérieure) qui définit un attribut `name` et une méthode abstraite (`sound`) que les classes enfants doivent implémenter.
- La classe `Mammal` hérite de `Animal` et ajoute un attribut `has_fur` et une méthode `sound` qui est spécifique aux mammifères.
- Les classes `Dog` et `Cat` sont des classes-enfants (ou sous-classes) de `Mammal`. Elles héritent de tous les attributs et méthodes de `Mammal`, ainsi que de la méthode `sound` implémentée dans `Mammal`.
- Les instances de `Dog` et `Cat` peuvent appeler la méthode `sound` qui est spécifique à leur classe, mais elles peuvent également utiliser les méthodes et attributs hérités de `Mammal`.

Voici un exemple d'utilisation :

```

1  python
2  my_dog = Dog()
3  my_cat = Cat()
4
5  print(my_dog.name) # Fido

```

```
6 print(my_cat.name)  # Whiskers
7
8 my_dog.sound()    # Fido aboi.
9 my_cat.sound()   # Whiskers miaou.
10
11 print(my_dog.has_fur)  # True (héritage de Mammal)
12 print(my_cat.has_fur)  # False (modification dans Cat)
13
```