|  |
| --- |
|  |
| Python |
|  |

|  |
| --- |
| Soubika BISOO  13/12/2021 |

Table des matières

[Veille 1 : Les variables 3](#_Toc91608403)

[Qu’est-ce qu’une variable ? 3](#_Toc91608404)

[Lister les types de variables : 3](#_Toc91608405)

[Lister les opérateurs Python : 3](#_Toc91608406)

[Lister les fonctions utiles pour traiter les chaînes de caractères 3](#_Toc91608407)

[Veille 2 : Les listes et les Tuples 4](#_Toc91608408)

[Qu’est-ce qu’une liste en Python ? 4](#_Toc91608409)

[Lister les fonctions utiles pour manipuler les listes 4](#_Toc91608410)

[Comment parcourir une liste ? 4](#_Toc91608411)

[Explorer les listes de compréhension 4](#_Toc91608412)

[Veille 3 : Les Dictionnaires 6](#_Toc91608413)

[Qu’est-ce qu’un dictionnaire en Python ? 6](#_Toc91608414)

[Comment créer un dictionnaire en Python ? 6](#_Toc91608415)

[Lister les opérations communes pour manipuler les dictionnaires 6](#_Toc91608416)

[Comment parcourir un dictionnaire ? 6](#_Toc91608417)

[Manipuler un dictionnaire de listes 7](#_Toc91608418)

[Manipuler un dictionnaire de dictionnaires 7](#_Toc91608419)

[Veille 3 : La Programmation Orienté Objet (POO) 9](#_Toc91608420)

[Programmation orienté objet versus procédural ? 9](#_Toc91608421)

[Qu’est-ce qu'une classe ? 11](#_Toc91608424)

[Qu’est-ce qu'un constructeur ? 11](#_Toc91608425)

[Qu’est-ce qu’un attribut ? 11](#_Toc91608426)

[Qu’est-ce que l'encapsulation ? 11](#_Toc91608427)

[Accesseurs et mutateurs ? 12](#_Toc91608428)

[Qu’est-ce que l'héritage ? 12](#_Toc91608429)

# Veille 1 : Les variables

## Qu’est-ce qu’une variable ?

Zone de la mémoire de l’ordinateur dans laquelle une valeur est stockée. Aux yeux du programmeur, cette variable est définie par un nom, alors que pour l’ordinateur, il s’agit d’une adresse, cad d’une zone particulière de la mémoire

## Lister les types de variables :

* Char
* Integer int
* Floats
* String str
* Boleen

## Lister les opérateurs Python :

* Arithmétiques 🡺 +, -, \*, /, %, \*\*, //
* Relationnels 🡺 == != < > <= >=
* Logiques 🡺 and or not
* Affectation 🡺 = += -= \*=
* Spéciaux 🡺 is, is not, in, not in

## Lister les fonctions utiles pour traiter les chaînes de caractères

* Concaténation 🡺 +
* .format()
* .find()
* Type()
* .lower()
* .capitalize()
* .title()
* .replace()
* .translate
* .split()

# Veille 2 : Les listes et les Tuples

## Qu’est-ce qu’une liste en Python ?

Une liste est une structure de données qui contient une série de valeurs. Python autorise la construction de liste contenant des valeurs de types différents (par exemple entier et chaîne de caractère), ce qui leur confère une grande flexibilité. Une liste est déclarée par une série de valeurs séparées par des virgules, encadré par des crochets.

## Lister les fonctions utiles pour manipuler les listes

* append()
* len()
* list(range())
* min()
* max()
* sum()

## Comment parcourir une liste ?

Elément par élément : liste[indice]

|  |
| --- |
| Boucle For :  for element in liste :  print(element) |

## Explorer les listes de compréhension

La notion de liste de compréhension représente une manière originale et très puissante de générer des listes. La syntaxe de base consiste au moins en une boucle for au sein de crochets précédés d’une variable.

new\_list = [function(item) **for** item **in** list **if** condition(item)]

Ex :

>>> [i for i in range (10)]

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> [2 for i in range (10)]

[2, 2, 2, 2, 2, 2, 2, 2, 2, 2]

List VS Tuples

Les tuples correspondent aux listes à la différence qu’ils sont non modifiables. Ils utilisent les parenthèses au lieu des crochets

# Veille 3 : Les Dictionnaires

## Qu’est-ce qu’un dictionnaire en Python ?

Les dictionnaires sont des collections non ordonnées d’objet, c'est-à-dire qu’il n’y a pas de notion d’ordre (donc pas d’indice). On accède aux valeurs d’un dictionnaire par des clés.

## Comment créer un dictionnaire en Python ?

La définition d’un dictionnaire vide se fait avec les accolades {}.

Ensuite, on remplit le dictionnaire avec différentes clés auxquelles on affecte des valeurs.

On peut mettre autant de clés qu’on veut dans un dictionnaire.

Exemple :

Une image contenant texte

Description générée automatiquement

Une image contenant texte

Description générée automatiquement

## Lister les opérations communes pour manipuler les dictionnaires

* Ajouter des clés et des valeurs
* Modifier les valeurs des clés
* Supprimer des clés valeurs
* Modifier les clés 🡺 plus compliquer

## Comment parcourir un dictionnaire ?

|  |
| --- |
| Boucle For :  for key, value in my\_dict.items() :  print(‘{} - {}’.format(key, value) |

|  |
| --- |
| Méthode .items()  Print(my\_dict.items()) |

## Manipuler un dictionnaire de listes

Il est possible d’avoir une liste à la place des valeurs. On obtient une structure qui ressemble à une base de données :

|  |
| --- |
| favorite\_numbers = {'eric': [3, 11, 19, 23, 42],                      'ever': [2, 4, 5],                      'willie': [5, 35, 120],                      }  for k in favorite\_numbers:  print("{} : {}".format(k, favorite\_numbers[k]))  eric : [3, 11, 19, 23, 42]  ever : [2, 4, 5]  willie : [5, 35, 120] |

## Manipuler un dictionnaire de dictionnaires

|  |
| --- |
| pets = {'willie': {'kind': 'dog', 'owner': 'eric', 'vaccinated': True},  'walter': {'kind': 'cockroach', 'owner': 'eric', 'vaccinated': False},  'peso': {'kind': 'dog', 'owner': 'chloe', 'vaccinated': True},  }  for k in pets.keys():  print("name : {}".format(k))  for k2, v2 in pets[k].items() :  print("\t{} : {}".format(k2, v2))  name : willie  kind : dog  owner : eric  vaccinated : True  name : walter  kind : cockroach  owner : eric  vaccinated : False  name : peso  kind : dog  owner : chloe  vaccinated : True |

# Veille 3 : La Programmation Orienté Objet (POO)

## Programmation orienté objet versus procédural ?

La différence :

* Programmation procédurale : les programmes sont basés sur des fonctions, et les données peuvent être facilement accessible et modifiable
* Programmation orientée objet : chaque programme est constitué d’entités appelées objets, qui ne sont pas facilement accessible et modifiables

Programmation procédurale :

## 

Dans la programmation procédurale, le programme est divisé en petites parties appelées procédures ou fonctions. Comme son nom l’indique, la programmation procédurale contient une procédure étape par étape à exécuter. Ici, les problèmes sont décomposés en petite parties et, pour résoudre chaque partie, une ou plusieurs fonctions sont utilisées.

Programmation orienté objet :

## 

Dans la programmation orientée objet, le programme est divisé en parties appelées objets. La programmation orientée objet est un concept de programmation qui se concentre sur l’objet plutôt que sur les actions et les données plutôt que sur la logique

|  |  |  |
| --- | --- | --- |
|  | Programmation Procédurale | Programmation Orientée Objet |
| Programmes | Le programme principal est divisé en petites parties selon les fonctions. | Le programme principal est divisé en petit objet en fonction du problème. |
| Les données | Chaque fonction contient des données différentes. | Les données et les fonctions de chaque objet individuel agissent comme une seule unité. |
| Permission | Pour ajouter de nouvelles données au programme, l’utilisateur doit s’assurer que la fonction le permet. | Le passage de message garantit l’autorisation d’accéder au membre d’un objet à partir d’un autre objet. |
| Exemples | Pascal, Fortran | PHP5, C ++, Java. |
| Accès | Aucun spécificateur d’accès n’est utilisé. | Les spécificateurs d’accès public, *private*, et *protected* sont utilisés. |
| La communication | Les fonctions communiquent avec d’autres fonctions en gardant les règles habituelles. | Un objet communique entre eux via des messages. |
| Contrôle des données | La plupart des fonctions utilisent des données globales. | Chaque objet contrôle ses propres données. |
| Importance | Les fonctions ou les algorithmes ont plus d’importance que les données dans le programme. | Les données prennent plus d’importance que les fonctions du programme. |
| Masquage des données | Il n’y a pas de moyen idéal pour masquer les données. | Le masquage des données est possible, ce qui empêche l’accès illégal de la fonction depuis l’extérieur. |

Les failles de la programmation procédurale posent le besoin de la programmation orientée objet. La programmation orientée objet corrige les défauts de la programmation procédurale en introduisant le concept d’*objet* et de *classe.* Il améliore la sécurité des données, ainsi que l’initialisation et le nettoyage automatique des objets. La POO permet de créer plusieurs instances de l’objet sans aucune interférence.

## Qu’est-ce qu'une classe ?

Une classe déclare des propriétés communes à un ensemble d’objet. La classe déclare des attributs représentant l’état des objets et des méthodes représentant leur comportement.

Une classe représente donc une catégorie d’objets. Il apparaît aussi comme une usine à partir de laquelle il est possible de créer des objets. On parle alors d’un objet en tant qu’instance d’une classe (création d’un objet ayant les propriétés de la classe).

## Qu’est-ce qu'un constructeur ?

Si lors de la création d’un objet nous voulons qu’un certain nombre d’action soit réalisées, nous pouvons utiliser un constructeur.

Un constructeur n’est rien d’autre qu’une méthode, sans valeur de retour, qui porte un nom imposé par le langage python : \_\_init\_\_(). Cette méthode sera appelée lors de la création de l’objet. Le constructeur peut disposer d’un nombre quelconque de paramètre, éventuellement aucun.

## Qu’est-ce qu’un attribut ?

En programmation, les attributs sont des entités qui définissent les propriétés d’objets, d’éléments, ou de fichiers. Les attributs sont habituellement composés d’un identificateur (ou nom ou clé) et d’une valeur.

Il existe 3 types d’attributs :

* Les attributs **d’instance** : variables définies à l’aide de *self.* Elles sont relatives à l’instance, et ne peuvent être accédées dans instanciation. Dans le cadre des méthodes, ce sont les méthodes classiques d’une classe, qui possèdent *self* en premier paramètre
* Les attributs de **classe** : variables définies directement dans le corps de la classe. Elles peuvent être accédées par la classe, sans passer l’instanciation. Les attributs de classe peuvent se référencer entre eux, mais ne peuvent pas accéder aux attributs d’instance
* Les attributs **statiques** : attributs qui n’ont pratiquement aucun lien avec la classe. Seules les méthodes peuvent être statiques, et l’ajout par rapport aux attributs de la classe est minime : on a plus besoin de spécifier le paramètre *cls*. Pour créer un attribut statique, il suffit de faire précéder la méthode pas *@staticmethod*

## Qu’est-ce que l'encapsulation ?

L’encapsulation permet d’enfermer dans une capsule les données brutes afin d’éviter des erreurs de manipulation ou de corruption des données. Permet de cacher des méthodes et des attributs à l’extérieur de la classe.

Il permet d’éviter une modification par erreur des données d’un objet. En effet, il n’est alors pas possible d’agir directement sur les données d’un objet ; il est nécessaire de passer par ses méthodes qui jouent le rôle d’interface obligatoire.

## Accesseurs et mutateurs ?

Parmi les différentes méthodes que comporte une classe, on a souvent tendance à distinguer :

* Les constructeurs
* Les accesseurs qui fournissent des informations relatives à l’états d’un objet, c’est-à-dire aux valeurs de certains de ses attributs (généralement privés) sans les modifier
* Les mutateurs qui modifient l’état d’un objet, donc les valeurs de certains de ses attributs

## Qu’est-ce que l'héritage ?

Le concept d’héritage signifie qu’une classe B va hériter des mêmes attributs et méthodes qu’une classe A. Lorsqu’une instance de la classe B est créée, on peut alors appeler les méthodes présentes dans la classe A par la classe B.

Dans ce cas, on crée une nouvelle classe B liée à la classe A et qui ajoute de nouvelles propriétés :

* A est une *généralisation* de B et B est une *spécialisation* de A
* A est une *super-classe* de B et B est une *sous-classe* de A
* A est la *classe mère* de B et B est une *classe fille* de A

Lorsqu’une classe hérite des méthodes d’une classe parent, il est possible de surcharger une méthode, qui consiste à redéfinir la méthode de la classe parent pour que les deux classes ne fassent pas les mêmes tâches : c’est ce qu’on appelle le polymorphisme.