Intelligence Artificielle 1

Cours 2 et 3 - Introduction Python

Table des matières

Syntax (indentation)

Python fonctionne sur la base de l'indentation pour la structuration ainsi que la validation du code.

Contrairement aux autres langages populaires, une mauvaise indentation/structure du code va causer une erreur de compilation ou des mauvaises exécutions.

*S'applique aussi aux fonctions.

Syntax (indentation)

Listing: https://www.w3schools.com/python/python_syntax.asp

```
1  # Steve Levesque, All rights reserved
2
3  # Incorrect
4  if 2 > 1:
5  print("2 is larger than 1 : True")
6
7  # Correct
8  if 2 > 1:
9      print("2 is larger than 1 : True")
```

Typage et casting

Le type est enforcé en Python. Alors il faut bien les gérer lorsque l'on programme.

Généralement, les langages sont typés lorsque leur but demande beaucoup de précision.

Par exemple, dans un langage typé, il n'est pas possible de concaténer un chiffre (int, float, etc.) avant de l'avoir converti en chaîne de caractères (string).

Typage et casting

Listing: https://www.w3schools.com/python/python_datatypes.asp

```
1  # Steve Levesque, All rights reserved
2
3  """
4      print("La note de passage est : " + 60)
5            ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
6  TypeError: can only concatenate str (not "int") to str
7  """
8  print("La note de passage est : " + 60)
```

Typage et casting

Listing: https://www.w3schools.com/python/python_datatypes.asp

```
1  # Steve Levesque, All rights reserved
2
3  """
4  La note de passage est : 60
5  """
6  print("La note de passage est : " + str(60))
```

Variables et structures de données - Tableaux

Les listes sont utilisées pour stocker plusieurs éléments dans une seule variable.

Les listes sont l'un des 4 types de données intégrées en Python utilisées pour stocker des collections de données, les 3 autres sont **Tuple**, **Set** et **Dictionary**, tous avec des qualités et des utilisations différentes.

Les listes sont créées à l'aide de crochets carrés : "[]"

Variables et structures de données - Tableaux

Listing: https://www.w3schools.com/python/python_lists.asp

```
1  # Steve Levesque, All rights reserved
2
3  # Base declaration
4  grades = [60, 80, 90, 80]
5
6  # Print directly
7  print(grades)
8
9  # Convert to string and print with concat to another string sentence
10 print("The array itself as a string : " + str(grades))
11
12 # Iterative print for each element in array (in order)
13 print()
14 print("Print each element of the array with a special loop : ")
15 for grade in grades:
16     print(grade)
```

Variables et structures de données - Dictionnaires

Les dictionnaires sont utilisés pour stocker les valeurs de données dans des paires clé/valeur.

Un dictionnaire est une collection ordonnée*, modifiable et qui n'autorise pas les doublons.

Les dictionnaires sont écrits avec des accolades "{}", et ont des clés et des valeurs.

*Depuis la version 3.7 de Python, les dictionnaires sont ordonnés. Dans Python 3.6 et les versions antérieures, les dictionnaires ne sont pas ordonnés.

Variables et structures de données - Dictionnaires

Listing: https://www.w3schools.com/python/python_dictionaries.asp

```
1  # Steve Levesque, All rights reserved
2
3  # "key": "value"
4  example =      {
5    "brand": "Ford",
6    "electric": False,
7    "year": 1964,
8    "year": 2020, # Will replace 1964 with 2020 since duplicate
9    "colors": ["red", "white", "blue"]
10 }
11
12 # {'brand': 'Ford', 'electric': False, 'year': 2020,
13 # 'colors': ['red', 'white', 'blue']}
14 print(example)
```

Fonctions

Une fonction est un bloc de code qui ne s'exécute que lorsqu'elle est appelée.

Vous pouvez transmettre des données, appelées paramètres, à une fonction.

Une fonction peut renvoyer des données en conséquence.

Composantes d'une fonction :

- Appels de fonction
- Les paramètres (variables listées) et les arguments (les valeurs réelles passées à la fonction) :
  - Le nombre de paramètres/arguments
  - Les valeurs par défaut
  - Le type du paramètre

Appels de fonction

Listing: https://www.w3schools.com/python/python_functions.asp

```
1  # Steve Levesque, All rights reserved
2
3  # Declare the function
4  def addition():
5      print("Simple function call without values...")
6
7
8  # Call the function to execute the code inside it.
9  addition()
```

Les paramètres et arguments

Listing: https://www.w3schools.com/python/python_functions.asp

```
1  # Steve Levesque, All rights reserved
2
3  # Declare the function
4  # The default values of "first" and "second" are equal 1 when no arguments
5  def addition(first=1, second=1):
6      # Return gives the result back to the function call.
7      return first + second
8
9
10 # Call the function to execute the code inside it.
11 # When no values (arguments) are passed, the parameters
12 # are by default from the specified values in the function declaration.
13 print(addition())
```

Les paramètres et arguments

Listing: https://www.w3schools.com/python/python_functions.asp

```
1  # Steve Levesque, All rights reserved
2
3  # Declare the function
4  # The default values of "first" and "second" are equal 1 when no arguments
5  def addition(first=1, second=1):
6      # Return gives the result back to the function call.
7      return first + second
8
9
10 # Call the function to execute the code inside it.
11 # With values (args) specified, the function will yield the result.
12 # "5"
13 print(addition(2, 3))
```

Les paramètres et arguments

```python
# Steve Levesque, All rights reserved

"""
addition
Return the added total of an array of numbers.

numbers [List]: A list of numbers
"""
def addition(numbers):
    res = 0

    for num in numbers:
        res += num

    return res

# Call the function to execute the code inside it.
nums = [1, 3, 5]
print(addition(nums))
```

Documentation du code (commentaires)

En Python la documentation est simple et rapide.

Il y a deux types de commentaires :

- Par ligne (unitaire) : avec le “hashtag” (#)
- Par bloc (multiple : avec les guillemets simples ou doubles en suite de 3 (”’ ou ”””)

Documentation du code (commentaires)

Listing: https://www.w3schools.com/python/python_comments.asp

```
1  # Steve Levesque, All rights reserved
2
3  """
4  A comment in block
5  1
6  2
7  """
8
9  # Comment for singular lines
10 # 1
11
12 """
13     This is a wrong comment evaluated as code...
14               ^^^^^
15 SyntaxError: invalid syntax
16 """
17 This is a wrong comment evaluated as code...
```

Importation de fonction dans des fichiers

Pour organiser le code dans un gros projet et/ou faire des tests unitaires, il est très pertinent de savoir comment importer des fonctions d'un fichier externe dans un nouveau fichier.

Importation de fonction dans des fichiers

```
1  # Steve Levesque, All rights reserved
2
3  def addition(numbers):
4      res = 0
5
6      for num in numbers:
7          res += num
8
9      return res
```

Importation de fonction dans des fichiers

```
1  # Steve Levesque, All rights reserved
2
3  #    filename          function name
4  from calculator import addition
5
6
7  # Import the function from the file and use it as it is
8  # "11"
9  print(addition([3, 6, 1, 1]))
```

Exercice formatif 2

Voir l'énoncé dans la remise sur Léa.

Erreurs (compilation et logique)

Puisque le code est écrit par des humains (ou maintenant par l'intelligence artificielle), celui-ci est sujet aux erreurs.

Il est possible de s'équiper avec des méthodes pour empêcher deux types d'erreurs courantes :

- Compilation : le code (une partie) n'exécute pas du tout et on voudrait quand même continuer l'exécution du programme
  - Solution : Exceptions
- Logique : le code exécute, mais le retour voulu est erroné
  - Solution : Tests unitaires

Exceptions (compilation)

En Python, il est possible de gérer les exceptions avec “try” et “expect”.

Les erreurs de compilation peuvent survenir de manière planifiée dans certains cas et seulement une gestion d’exception est la solution pour résoudre le problème.
Par contre, il est généralement non recommandé d’utiliser trop souvent un gestionnaire d’exception et de trouver une solution à la source lorsque possible.

Exceptions (compilation)

Listing: https://www.w3schools.com/python/python_comments.asp

```
1  # Steve Levesque, All rights reserved
2
3  # The variable "x" is not defined.
4  # In a normal execution, there would be a compilation error
5  # and the program would stop executing.
6  try:
7    print(x)
8  except:
9    print("An exception occurred")
10
11 # Result:
12 # An exception occurred
```

Tests unitaires (logique)

Si un programme retourne une valeur erronée, celui-ci compile tout de même. L'ordinateur (le compilateur) n'a pas de compréhension vis-à-vis la logique du code à son domaine d'application.

Tests unitaires (logique)

```python
# Steve Levesque, All rights reserved
import unittest
from calculator import addition

class TestCalculatorAddition(unittest.TestCase):

    def test_add_from_array(self):
        nums = [1, 3, 4]
        res = 8
        self.assertEqual(addition(nums), res)

    def test_add_from_array_large(self):
        nums = [1, 3, 4, 1, 1, 2, 3, 4, 5, 6, 7, 1, 3]
        res = 41
        self.assertEqual(addition(nums), res)

if __name__ == '__main__':
    unittest.main()
```

Tests unitaires (logique)

Si la fonction est mauvaise, voici un log avec les erreurs et de l'information additionnelle pertinente.

```
1  # Steve Levesque, All rights reserved
2  """
3  FAIL: test_add_from_array (__main__.TestCalculatorAddition.test_add_from_array)
4  ----------------------------------------------------------------------
5  Traceback (most recent call last):
6      self.assertEqual(addition(nums), res)
7  AssertionError: -8 != 8
8
9  ======================================================================
10 FAIL: test_add_from_array_large (__main__.TestCalculatorAddition.test_add_from_
11 ----------------------------------------------------------------------
12 Traceback (most recent call last):
13     self.assertEqual(addition(nums), res)
14 AssertionError: -41 != 41
15
16 ----------------------------------------------------------------------
17 Ran 2 tests in 0.001s
18
19 FAILED (failures=2)
20 """
```

Manipulation de structure de données de base (tableaux, dictionnaires)

Dans tous les langages, il est normalement possible de faire des opérations CRUD sur les structures de données.

Python supporte de manière très intuitive les opérations CRUD :

- (C)reate
- (R)ead
- (U)pdate
- (D)elete

Les tableaux et dictionnaires seront abordés.

Manipulation de structure de données de base - Tableaux

```python
# Steve Levesque, All rights reserved

crud_list = []

# [C]reate
crud_list.append(1)
print(crud_list)

# [R]ead
print(crud_list[0])

# [U]pdate
crud_list[0] = 42
print(crud_list)

# [D]elete
del crud_list[0]
print(crud_list)

# Results: [1] 1 [42] []
```

Manipulation de structure de données de base - Dictionnaires

```
1  # Steve Levesque, All rights reserved
2
3  crud_dict = {}
4
5  # [C]reate
6  crud_dict["banana"] = "red"
7  print(crud_dict)
8
9  # [R]ead
10 print(crud_dict["banana"])
11
12 # [U]pdate
13 crud_dict["banana"] = "yellow"
14 print(crud_dict)
15
16 # [D]elete
17 del crud_dict["banana"]
18 print(crud_dict)
19
20 # results: {'banana': 'red'} red {'banana': 'yellow'} {}
```

Utilisation de modules externes populaires "open-source" pour aider à la programmation en AI

Il suffit d'installer le module en suivant les étapes suivantes :

- Rechercher "pip install (nom du module, i.e. numpy)" sur votre navigateur favori.
- Cliquer sur le premier résultat qui apparait en lien avec le module (en norme) : le site `https://pypi.org/project/numpy/`
- Copier la commande d'installation sur le site : i.e. "pip install numpy" pour cet exemple
- Aller dans la console de votre IDE avec le .venv d'activé
- Écrire la commande copiée précédemment, attendre, et le module s'installe tout seul

NB : Il est pertinent de faire comme ceci puisque les noms des modules ne sont pas toujours simples et intuitifs dans le registre pip pour des raisons diverses que l'auteur peut avoir eu à considérer.

Exemple 1 - Module Numpy

```
1  # Steve Levesque, All rights reserved
2
3  """
4  (.venv) PS D:Steve> pip install numpy
5  Collecting numpy
6    Obtaining dependency information for numpy from https://files.pythonhosted.org
7    Downloading numpy-1.26.4-cp312-cp312-win_amd64.whl.metadata (61 kB)
8       ---------------------------------------- 61.0/61.0 kB 819.4 kB/s eta 0:00:0
9  Downloading numpy-1.26.4-cp312-cp312-win_amd64.whl (15.5 MB)
10     ---------------------------------------- 15.5/15.5 MB 3.6 MB/s eta 0:00:00
11 Installing collected packages: numpy
12 Successfully installed numpy-1.26.4
13
14 [notice] A new release of pip is available: 23.2.1 -> 24.0
15 [notice] To update, run: python.exe -m pip install --upgrade pip
16 """
```

Module - Numpy

```
1  # Steve Levesque, All rights reserved
2
3  # The module installed earlier, made by experts and open-source
4  import numpy as np
5
6  arr = np.array([8, 7, 6, 1, 2, 3, 4, 5])
7  arr = np.sort(arr)
8
9  # [1 2 3 4 5 6 7 8]
10 print(arr)
```

Exemple 2 - Module Sklearn

Listing:
https://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html

```
1  # Author: Gael Varoquaux <gael dot varoquaux at normalesup dot org>
2  # License: BSD 3 clause
3  from sklearn import datasets, metrics, svm
4  from sklearn.model_selection import train_test_split
5
6  # MNIST Project : load database and flatten images
7  digits = datasets.load_digits()
8  data = digits.images.reshape((len(digits.images), -1))
9
10 # Create a classifier: a support vector classifier
11 clf = svm.SVC(gamma=0.001)
12 # Split data into 50% train and 50% test subsets
13 X_train, X_test, y_train, y_test = train_test_split(
14     data, digits.target, test_size=0.5, shuffle=False
15 )
16
17 # Learn the digits on the train subset
18 clf.fit(X_train, y_train)
19 # Predict the value of the digit on the test subset
20 predicted = clf.predict(X_test)
```

Exercice formatif 3

Voir l'énoncé dans la remise sur Léa.

Bibliographie

- https://www.w3schools.com/python/
- https://pypi.org/project/pip/
- https://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html