PYTHON

Présenté par :

Xavier TABUTEAU

Règles de nommage

- Python ne possède pas de syntaxe particulière pour créer ou déclarer une variable.

- Il existe quelques règles usuelles pour la dénomination des variables.

- Les variables vont pouvoir stocker différents types de valeurs comme des nombres, des chaînes de caractères, des booléens, et plus encore.

- La casse est significative dans les noms de variables.

Il y a 3 règles à respecter pour déclarer une variable :

- Le nom doit débuter par une lettre ou un underscore « _ ».
- Le nom d'une variable doit contenir que des caractères alphanumériques courant, sans espace ni caractères spéciaux ou accents.
- On ne peut pas utiliser certains mots qui possède déjà une signification en Python. Ce sont les mots réservés.

Les types

- Integer

```
>>> a = 15
>>> type(a)
>>> class <int>
```

- String

```
>>> texte1 = 'Les crêpes.'
>>> texte2 = ' "Oui", répondit-il, '
>>> texte3 = "j'aime bien."
>>> print(texte1, texte2, texte3)
>>> 'Les crêpes. Oui, répondit-il, j’aime bien. ‘
```

- Float

```
>>> b = 16.0
>>> type(b)
>>> class <Float>
```

- Boolean

```
>>> c = True
>>> type(c)
>>> class <bool>
```

Les chaînes de caractères

Une donnée de type string peut se définir en première approximation comme une suite quelconque de caractères. Dans un script python, on peut délimiter une telle suite de caractères, soit par des apostrophes (simple quotes), soit par des guillemets (double quotes).

- **Le caractère spécial « \ » (antislash)**

En premier lieu, il permet d'écrire sur plusieurs lignes une commande qui serait trop longue pour tenir sur une seule (cela vaut pour n'importe quel type de commande).
À l'intérieur d'une chaîne de caractères, l'antislash permet d'insérer un certain nombre de codes spéciaux (sauts à la ligne, apostrophes, guillemets, etc.).

Exemples :

```
>>> print("ma\tpetite\nchaine")        # affiche ma        petite
                                       # chaine
```

Notion de formatage

- Format historique (Python 1):

```
%s    : string                  >>> n = "Celine"
%d    : decimal integer         >>> a = 42
%f    : float                   >>> print("nom : %s - age : %d" % (n, a))
%g    : generic number          >>> print("nom : %(nom)s - age : (age)s"  % {"nom": n, "age": a}
```

Ce format vient directement de la fonction printf du langage C.

- Depuis Python 2.6 :

```
.format()                       >>> print("nom : {1} - age : {0}".format(a, n))
                                >>> print("nom : {age} - age : {nom} ".format(nom=n, age=a))
```

- Depuis Python 3.6 :

```
fstring                         >>> print(f"nom : {n} - age : {a}")
```

Les 3 formatages existent dans les dernières versions de Python.

Typage dynamique fort

Python est fortement typé dynamiquement.

Un typage fort signifie que le type d'une valeur ne change pas de manière inattendue. Chaque changement de type nécessite une conversion explicite.

En Python, le typage dynamique signifie que le type des variables est déterminé automatiquement au moment de l'exécution (et non lors de l'écriture du code). En d'autres termes, il n'y a pas besoin de déclarer le type d'une variable à l'avance, Python le déduit automatiquement en fonction de la valeur qu'on lui assigne. Les valeurs attribuées n'ont pas de limite contrairement aux autres langages. La seule limite est la mémoire du PC.

```
>>> points = 3.2                                # points est du type float (nombre décimal)
>>> print("Tu as " + points + " points !")      # Génère une erreur de typage
>>> points = int(points)                        # points est maintenant du type int (entier),
                                                # sa valeur est arrondie à l'unité inférieure (ici 3)
>>> print("Tu as " + points + " points !")      # Génère une erreur de typage
>>> points = str(points)                        # points est maintenant du type str (string)
>>> print("Tu as " + points + " points !")      # Plus d'erreur de typage, affiche "Tu as 3 points !"
```

Variables

Les collections

Les chaînes de caractères que nous avons abordées constituaient un premier exemple de données composites. On appelle ainsi les structures de données qui sont utilisées pour regrouper de manière structurée des ensembles de valeurs.

Listes

```
#Déclarer une list avec la possibilité de modifier son contenu
list_data = [1, 2, 3, 4]
```

Tuples

```
 #Déclarer un tuple sans la possibilité de modifier son contenu
 tuple_data = (1, 2, 3, 4)
```

Sets

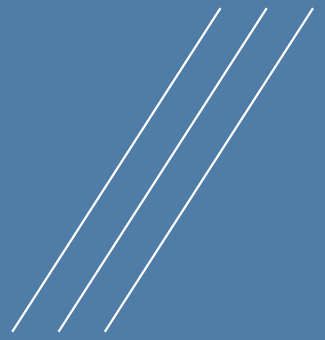
```
# Un set ne contient pas de doublon.
# ici, le deuxième élément "pierre" sera ignoré.
mon_set = {"pascal", "pierre", "paul", "pierre"}
```

Dictionnaires

```
#Construire un dictionnaire
dico = {}
dico['computer'] = 'ordinateur'
dico['mouse'] ='souris'
dico['keyboard'] ='clavier'
print("dico")
dico = {'computer': 'ordinateur', 'keyboard': 'clavier', 'mouse': 'souris'}
```

collection.py
ex_collection

PYTHON



Présenté par

Xavier TABUTEAU