PYTHON PAR LA PRATIQUE

Présenté par :

Xavier TABUTEAU

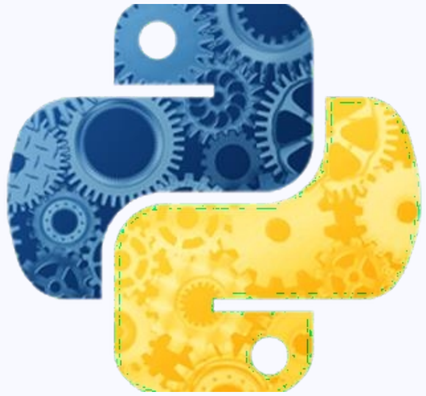
03. Variables

Python ne possède pas de syntaxe Particulière pour créer ou “declarer” une variable

Il existe quelques règles usuelles pour la Denomination des variables

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

la casse est significative dans les noms de variables



Python: Règle de Nommage

REGLE 1

Le nom doit commencer par une lettre ou par un underscore.

REGLE 2

Le nom d'une variable ne doit contenir que des caractères alphanumériques courants (pas d'espace dans le nom d'une variable ni de caractères spéciaux comme des caractères accentués ou tout autre signe).

REGLE 3

On ne peut pas utiliser certains mots qui possèdent déjà une signification spéciale pour le langage (on parle de mots "réservés").

https://fr.wikibooks.org/wiki/Programmation_Python/Tableau_des_mots_r%C3%A9serv%C3%A9s

03. Variables

Les Types

Integer

```
>>> a = 15
>>> type(a)
```

String

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

Float

```
>>> b = 16.0
>>> type(b)
```

Booleen

```
>>> c = True
>>> type(c)
```

03. Variables

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

%s: string
%d: decimal integer
%f: float
%g: generic number
.format()

```
>>> n = "Celine"
>>> a = 42
>>> print("nom : %s - age %d" %(n, a))
>>> print("nom : {1} - age : {0}".format(n, a))
>>> print(f"nom : {n} - age : {a}")
```

03. Variables

variables.py

Typage dynamique fort

Python est fortement typé dynamiquement.

Un typage fort signifie que le type d'une valeur ne change pas de manière inattendue. Une chaîne contenant uniquement des chiffres ne devient pas par magie un nombre, comme cela peut arriver en Perl. Chaque changement de type nécessite une conversion explicite.

Le typage dynamique signifie que les objets d'exécution (valeurs) ont un type qui peut être changer, par opposition au typage statique où les variables ont un type inchangeable.

```
>>> 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 !"
```

03. Variables

collection.py
Exercice 1

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'}
```

PYTHON INITIATION

Présenté par
Xavier TABUTEAU