Python

Python est un langage de programmation. Il est l'un des langages de programmation les plus intéressants du moment. Facile à apprendre, python est souvent utilisé en exemple lors de l'apprentissage de la programmation.

Python c'est quoi ?

Python est un langage de programmation inventé par Guido van Rossum. La première version de python est sortie en 1991.

Python est un langage de programmation interprété, c'est à dire qu'il n'est pas nécessaire de le compiler avant de l'exécuter. Si vous avez déjà touché un peu à la programmation, vous verrez que ce langage possède une certaine poésie. Les programmeurs s'amusent souvent à trouver la manière la plus jolie/efficace d'écrire une suite d'instructions. Rares sont ceux qui critiquent la logique Python -contrairement à javascript par exemple-.

Que fait Python ?

Python est à la fois simple et puissant, il vous permet d'écrire des scripts très simples mais grâce à ses nombreuses bibliothèques, vous pouvez travailler sur des projets plus ambitieux.

- Web: Aujourd'hui python combiné avec le framework Django est un très bon choix technologique pour des gros projets de sites internet.
- Système: Python est également souvent utilisé par les admin système pour créer des tâches dites répétitives ou simplement de maintenance. D'ailleurs si vous voulez créer des applications java en codant en python, c'est possible grâce au projet Jython.

Pourquoi préférer Python aux autres langages ?

Python est un langage facile à apprendre et son code est plus lisible, il est donc plus facile à maintenir. Il est parfois jusqu'à 5 fois plus concis que le langage Java par exemple, ce qui augmente la productivité du développeur et réduit mécaniquement le nombre de bugs.

Python est également utilisé dans les milieux scientifiques, par exemple la bioinformatique. Des librairies sont disponibles pour ce domaine comme le module biopython.

Il existe également des bibliothèques facilitant la création de jeux vidéo en 2D (et 3D) exemple: pyGame

Qui utilise Python ?

Google (Guido van Rossum a travaillé pour Google de 2005 à 2012), Yahoo, Microsoft, la Nasa revendique l'utilisation de Python, pour ne citer qu'eux.

Installation:

Installer python sur Linux ou MacOS

Si vous travaillez dans un environnement Linux ou MacOS, bonne nouvelle Python déjà installé.

Installer python sur Windows

Si vous êtes sur Windows, changez de système d'exploitation...

Personnellement j'adore travailler avec Ubuntu, en tant que développeur vous avez vraiment l'impression de maîtriser votre machine et en plus tout y est gratuit. De plus sous serez souvent amené à migrer votre travail sur un serveur de production qui peut tourner sur une distrib linux. Il faut savoir qu'aujourd'hui plus de projets sont hébergés sur des serveur linux que sur des licences propriétaires.

Mais si vous voulez tout de même garder Windows -malgré tous mes efforts pour vous convertir dans le monde libre-, vous pouvez télécharger un fichier d'installation python à cette adresse: Télécharger Python

Quelle version choisir ?

Essayez de prendre la version la plus récente / stable. À noter que la version la plus utilisée aujourd'hui est la version 2.7

Il existe des problèmes des compatibilités entre la version python 2 et 3. Je vous conseille donc d'apprendre python 2 puis d'apprendre les différences entre ces deux versions. Vous serez ainsi capable de gérer les problèmes liés à d'inévitables migrations. _____

Editeurs Python:

Sublime Text

Alors lui c'est mon petit favori. Il est ultra light, il est beau, il est fort, il est puissant!

Sublime text possède tout une panoplie de plugins dont vous serez vite accroc! Sa version de base est gratuite, une petite alerte vous demandera de temps en temps si vous voulez acheter une licence pour soutenir le projet mais rien ne vous oblige à le faire.

Pensez à installer le packagecontrol qui vous permettra d'installer les outils nécessaires à votre projet

Voici une petite liste des raccourcis le plus utiles:

```
Ctrl + X  Supprimer une ligne
Ctrl + P    Permet de naviguer dans n'importe quel fichier
Ctrl + R    Déplacer le curseur à une fonction du fichier en cours
Ctrl + L    Sélectionner la ligne en cours
Ctrl + D    Sélectionner le mot entier en cours
Ctrl + Shift + D    Dupliquer la ligne en cours
Ctrl + M    Diriger le curseur vers l'autre extrémité d'une fonction
Ctrl + G    Déplacer le curseur à la ligne X du fichier
Ctrl + Shift + T    Réouvrir le dernier fichier fermé
CTRL + SHIFT + F    Faire une recherche sur les fichiers d'un dossier
CTRL + ALT + P  Switcher de projet
```

L'édtieur Wing IDE

WingIDE est l'un des meilleurs éditeurs -avec interpréteur intégré- pour les débutants dans sa version gratuite.

WingIDE a été conçu par des développeurs python pour des développeurs python surtout pour l'enseignement de python. La version gratuite possède évidemment moins d'options que la version profesionnelle (le prix reste assez bas pour la version pro: 45 $).

Vous pouvez télécharger le logiciel dans sa version gratuite ici: IDE python WingIDE .

Si vous êtes sous Ubuntu téléchargez le .deb, double cliquez sur le fichier et cliquez sur installer. Si vous rencontrez des problèmes de dépendances pensez à exécuter la commande suivante:

```
sudo apt-get install -f
```

===

Calculs et variables:

Calculs

Une des premières fonctionnalités d'un interpréteur est de faire des calculs:

```
>>> 1+2
3
```

Vous pouvez ajouter des espaces, cela n'aura pas d'incidences:

```
>>> 1 + 2
3
```

Tous les opérateurs sont utilisables:

```
>>> 1-10
-9
>>> 2*10
20
>>> 100/4
25
>>> 10%4
2
>>> 2**3
8
```

La double étoile représente l'exposant.

Variables

Une variable est une sorte de boite virtuelle dans laquelle on peut mettre une (ou plusieurs) donnée(s). L'idée est de stocker temporairement une donnée pour travailler avec. Pour votre machine une variable est une adresse qui indique l'emplacement de la mémoire vive où sont stockées les informations que nous avons liées avec.

Affectons une valeur à la variable age que nous allons ensuite afficher:

```
>>> age = 30
>>> age
30
```

On va ensuite ajouter 10 à la valeur de cette variable:

```
>>> age = 30
>>> age = age + 10
>>> age
40
```

Il est possible de mettre une variable dans une autre variable.

```
>>> age = 30
>>> age2 = age
>>> age2
30
```

Vous pouvez mettre à peu près tout ce que vous voulez dans votre variable, y compris du texte:

```
>>> age = "J'ai 30 ans"
>>> age
"J'ai 30 ans"
```

Vous pouvez même multiplier une chaine de caractères:

```
>>> age = "jeune"
>>> age * 3
'jeunejeunejeune'
```

Evidemment, si vous essayez de faire des additions avec des variables qui sont des chiffres et d'autres qui sont du texte, l'interpréteur va vous gronder:

```
>>> age = "J'ai 30 ans"
>>> age
"J'ai 30 ans"
>>> age + 1
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
```

Vous remarquez que l'interpréteur est sympa puisqu'il vous dit ce qui ne va pas: Il ne peut pas concaténer `str` et `int`. _____

Nommer une variable

Vous ne pouvez pas nommer les variables comme bon vous semble, puisqu'il existe déjà des mots utilisés par Python. Voici la liste des mots réservé par python:

```
print in and or if del for is raise assert elif from lambda return break else global not try
```

Pourquoi ces mots sont-ils réservés? Parce qu'ils servent à faire autre chose. Nous verrons quoi plus en détail dans les prochains chapitres.

Pour nommer une variable vous devez obligatoirement utiliser les lettres de l'alphabet, les chiffres et le caractère "_" et "-". N'utilisez pas les accents, ni les signe de ponctuation ou le signe @. De plus les chiffres ne doivent jamais se trouver en première position dans votre variable:

```
>>> 1var = 1
File "<stdin>", line 1
1var = 1
   ^
SyntaxError: invalid syntax
```

Comme vous le remarquez, python refuse ce genre de syntaxe, mais il acceptera `var1 = 1.` ____

Les types de variables:

En python une variable est typée, c'est à dire qu'en plus d'une valeur, une variable possède une sorte d'étiquette qui indique ce que contient cette boite virtuelle.

Voici une liste de type de variable:

Les *integer* ou nombres entiers: comme son nom l'indique un entier est un chiffre sans décimales. Les *float* ou nombre à virgules: exemple : 1.5 Les *strings* ou chaine de caractères: pour faire simple tout ce qui n'est pas chiffre. Il en existe plein d'autres mais il est peut être encore un peu trop tôt pour vous en parler.

Pour connaitre le type d'une variable, vous pouvez utiliser la fonction `type()`

```
>>> v = 15
>>> type(v)
<type 'int'>

>>> v = "Olivier"
>>> type(v)
<type 'str'>

>>> v = 3.2
>>> type(v)
<type 'float'>
```

===

Les listes:

Les listes (ou `list` / `array`) en python sont une variable dans laquelle on peut mettre plusieurs variables.

Créer une liste en python

Pour créer une liste, rien de plus simple:

```
>>> liste = []
```

Vous pouvez voir le contenu de la liste en l'appelant comme ceci:

```
>>> liste
<type 'list'>
```

Ajouter une valeur à une liste python

Vous pouvez ajouter les valeurs que vous voulez lors de la création de la liste python:

```
>>> liste = [1,2,3]
>>> liste
[1, 2, 3]
```

Ou les ajouter après la création de la liste avec la méthode `append` (qui signifie "ajouter" en anglais):

```
>>> liste = []
>>> liste
[]
>>> liste.append(1)
>>> liste
[1]
>>> liste.append("ok")
>>> liste
[1, 'ok']
```

On voit qu'il est possible de mélanger dans une même liste des variables de type différent. On peut d'ailleurs mettre une liste dans une liste.

Afficher un item d'une liste

Pour lire une liste, on peut demander à voir l'index de la valeur qui nous intéresse:

```
>>> liste = ["a","d","m"]
>>> liste[0]
'a'
>>> liste[2]
'm'
```

Le premier item commence toujours avec l'index 0. Pour lire la premier item on utilise la valeur 0, le deuxième on utilise la valeur 1, etc.

Il est d'ailleurs possible de modifier une valeur avec son index

```
>>> liste = ["a","d","m"]
>>> liste[0]
'a'
>>> liste[2]
'm'
```

```
>>> liste[2] = "z"
>>> liste
['a', 'd', 'z']
```

Supprimer une entrée avec un index

Il est parfois nécessaire de supprimer une entrée de la liste. Pour cela vous pouvez utiliser la fonction `del`.

```
>>> liste = ["a", "b", "c"]
>>> del liste[1]
>>> liste
['a', 'c']
```

Inverser les valeurs d'une liste

Vous pouvez inverser les items d'une liste avec la méthode reverse.

```
>>> liste = ["a", "b", "c"]
>>> liste.reverse()
>>> liste
['c', 'b', 'a']
```

Compter le nombre d'items d'une liste

Il est possible de compter le nombre d'items d'une liste avec la fonction len.

```
>>> liste = [1,2,3,5,10]
>>> len(liste)
5
```

Compter le nombre d'occurences d'une valeur

Pour connaitre le nombre d'occurences d'une valeur dans une liste, vous pouvez utiliser la méthode count.

```
>>> liste = ["a","a","a","b","c","c"]
>>> liste.count("a")
3
>>> liste.count("c")
2
```

Trouver l'index d'une valeur

La méthode `index` vous permet de connaitre la position de l'item cherché.

```
>>> liste = ["a","a","a","b","c","c"]
>>> liste.index("b")
3
```

Manipuler une liste

Voici quelques astuces pour manipuler des listes:

```
>>> liste = [1, 10, 100, 250, 500]
>>> liste[0]
1
>>> liste[-1] # Cherche la dernière occurence
500
>>> liste[-4:] # Affiche les 4 dernières occurrences
[500, 250, 100, 10]
>>> liste[:] # Affiche toutes les occurences
[1, 10, 100, 250, 500]
>>> liste[2:4] = [69, 70]
[1, 10, 69, 70, 500]
>>> liste[:] = [] # vide la liste
[]
```

Boucler sur une liste

Pour afficher les valeurs d'une liste, on peut utiliser une boucle:

```
>>> liste = ["a","d","m"]
>>> for lettre in liste:
...     print lettre
...
a
d
m
```

Si vous voulez en plus récupérer l'index, vous pouvez utiliser la fonction `enumerate`.

```
>>> for lettre in enumerate(liste):
...     print lettre
...
(0, 'a')
(1, 'd')
(2, 'm')
```

Les valeurs retournées par la boucle sont des tuples.

Copier une liste

Beaucoup de débutants font l'erreur de copier une liste de cette manière

```
>>> x = [1,2,3]
>>> y = x
```

Or si vous changez une valeur de la liste y, la liste x sera elle aussi affectée par cette modification:

```
>>> x = [1,2,3]
>>> y = x
>>> y[0] = 4
>>> x
[4, 2, 3]
```

En fait cette syntaxe permet de travailler sur un même élément nommé différemment

Alors comment copier une liste qui sera indépendante?

```
>>> x = [1,2,3]
>>> y = x[:]
>>> y[0] = 9
>>> x
[1, 2, 3]
>>> y
[9, 2, 3]
```

Pour des données plus complexes, vous pouvez utiliser la fonction `deepcopy` du module `copy`

```
>>> import copy
>>> x = [[1,2], 2]
>>> y = copy.deepcopy(x)
>>> y[1] = [1,2,3]
>>> x
[[1, 2], 2]
>>> y
[[1, 2], [1, 2, 3]]
```

Transformer une string en liste

Parfois il peut être utile de transformer une chaine de caractère en liste. Cela est possible avec la méthode 'split.

```
>>> ma_chaine = "Olivier:ENGEL:Strasbourg"
>>> ma_chaine.split(":")
['Olivier', 'ENGEL', 'Strasbourg']
```

Transformer une liste en string

L'inverse est possible avec la méthode "join".

```
>>> liste = ["Olivier","ENGEL","Strasbourg"]
>>> ":".join(liste)
'Olivier:ENGEL:Strasbourg'
```

Trouver un item dans une liste

Pour savoir si un élément est dans une liste, vous pouvez utiliser le mot clé in de cette manière:

```
>>> liste = [1,2,3,5,10]
>>> 3 in liste
True
>>> 11 in liste
False
```

La fonction range

La fonction `range` génère une liste composée d'une simple suite arithmétique.

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Agrandir une liste par une liste

Pour mettre bout à bout deux listes, vous pouvez utiliser la méthode `extend`

```
>>> x = [1, 2, 3, 4]
>>> y = [4, 5, 1, 0]
>>> x.extend(y)
>>> print x
[1, 2, 3, 4, 4, 5, 1, 0]
```

Astuces

Afficher les 2 premiers éléments d'une liste

```
>>> liste = [1,2,3,4,5]
>>> liste[:2]
[1, 2]
```

Afficher le dernier item d'une liste:

```
>>> liste = [1, 2, 3, 4, 5, 6]
>>> liste[-1]
6
```

Afficher le 3ème élément en partant de la fin:

```
>>> liste = [1, 2, 3, 4, 5, 6]
>>> liste[-3]
4
```

Afficher les 3 derniers éléments d'une liste:

```
>>> liste = [1, 2, 3, 4, 5, 6]
>>> liste[-3:]
[4, 5, 6]
```

Vous pouvez additionner deux listes pour les combiner ensemble en utilisant l'opérateur +:

```
>>> x = [1, 2, 3]
>>> y = [4, 5, 6]
>>> x + y
[1, 2, 3, 4, 5, 6]
```

Vous pouvez même mutiplier une liste:

```
>>> x = [1, 2]
>>> x*5
[1, 2, 1, 2, 1, 2, 1, 2, 1, 2]
```

Ce qui peut être utile pour initialiser une liste:

```
>>> [0] * 5
[0, 0, 0, 0, 0]
```

Les tuples:

Un tuple est une liste qui ne peut plus être modifiée.

Créer un tuple

Pour créer un tuple, vous pouvez utiliser la syntaxe suivante:

```
>>> mon_tuple = ()
```

Ajouter une valeur à un tuple

Pour créer un tuple avec des valeurs, vous pouvez le faire de cette façon:

```
>>> mon_tuple = (1, "ok", "olivier")
```

Les parenthèses ne sont pas obligatoires mais facilite la lisibilité du code (rappelons que la force de python est sa simplicité de lecture):

```
>>> mon_tuple = 1, 2, 3
>>> type(mon_tuple)
<type 'tuple'>
```

Lorsque vous créez un tuple avec une seule valeur, n'oubliez pas d'y ajouter une virgule, sinon ce n'est pas un tuple.

```
>>> mon_tuple = ("ok")
>>> type(mon_tuple)
<type 'str'>
>>> mon_tuple = ("ok",)
>>> type(mon_tuple)
<type 'tuple'>
```

Afficher une valeur d'un tuple

Le tuple est une sorte de liste, on peut donc utiliser la même syntaxe pour lire les données du tuple.

```
>>> mon_tuple[0]
1
```

Et évidemment si on essaie de changer la valeur d'un index, l'interpreteur nous insulte copieusement:

```
>>> mon_tuple[1] = "ok"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

A quoi sert un tuple alors?

Le tuple permet une affectation multiple:

```
>>> v1, v2 = 11, 22
>>> v1
11
```

```
>>> v2
22
```

Il permet également de renvoyer plusieurs valeurs lors d'un appel d'une fonction:

```
>>> def donne_moi_ton_nom():
...     return ("olivier", "engel")
...
>>> donne_moi_ton_nom()
('olivier', 'engel')
```

On utilisera un tuple pour définir des sortes de constantes qui n'ont donc pas vocation à changer. _____

Les dictionnaires:

Un dictionnaire en python est une sorte de liste mais au lieu d'utiliser des index, on utilise des clés, c'est à dire des valeurs autres que numériques.

Comment créer un dictionnaire?

Pour initialiser un dictionnaire, on utile la syntaxe suivante:

```
>>> a = {}
```

Comment ajouter des valeurs dans un dictionnaire?

Pour ajouter des valeurs à un dictionnaire il faut indiquer une clé ainsi qu'une valeur:

```
>>> a = {}
>>> a["nom"] = "engel"
>>> a["prenom"] = "olivier"
>>> a
{'nom': 'engel', 'prenom': 'olivier'}
```

Vous pouvez utiliser des clés numériques comme dans la logique des listes.

Récupérer une valeur dans un dictionnaire

La méthode get vous permet de récupérer une valeur dans un dictionnaire et si la clé est introuvable, vous pouvez donner une valeur à retourner par défaut:

```
>>> data = {"name": "Olivier", "age": 30}
>>> data.get("name")
```

```
'Olivier'
>>> data.get("adresse", "Adresse inconnue")
'Adresse inconnue'
```

Vérifier la présence d'une clé dans un dictionnaire Vous pouvez utiliser la méthode haskey pour vérifier la présence d'une clé que vous cherchez:

```
>>> a.has_key("nom")
True
```

Supprimer une entrée de dictionnaire

Il est possible de supprimer une entrée en indiquant sa clé, comme pour les listes:

```
>>> del a["nom"]
>>> a
{'prenom': 'olivier'}
```

Récupérer les clés par une boucle

Pour récupérer les clés on utilise la méthode `keys`

```
>>> fiche = {"nom":"engel","prenom":"olivier"}
>>> for cle in fiche.keys():
...     print cle
...
nom
prenom
```

Récupérer les valeurs par une boucle

Pour cela on utilise la méthode `values`

```
>>> fiche = {"nom":"engel","prenom":"olivier"}
>>> for valeur in fiche.values():
...     print valeur
...
engel
olivier
```

Récupérer les clés et les valeurs par une boucle

Pour récupérer les clés et les valeurs en même temps, on utilise la méthode `items` qui retourne un tuple.

```
>>> fiche = {"nom":"engel","prenom":"olivier"}
>>> for cle,valeur in fiche.items():
...     print cle, valeur
...
nom engel
prenom olivier
```

Utiliser des tuples comme clé

Une des forces de python est la combinaison tuple/dictionnaire qui fait des merveilles dans certains cas comme lors de l'utilisation de coordonnées.

```
>>> b = {}
>>> b[(3,2)]=12
>>> b[(4,5)]=13
>>> b
{(4, 5): 13, (3, 2): 12}
```

Créer une copie indépendante d'un dictionnaire

Comme pour toute variable, vous ne pouvez pas copier un dictionnaire en faisant dic1 = dic2:

```
>>> d = {"k1":"olivier", "k2":"engel"}
>>> e = d
>>> d["k1"] = "XXX"
>>> e
{'k2': 'engel', 'k1': 'XXX'}
```

Pour créer une copie indépendante vous pouvez utiliser la méthode `copy`:

```
>>> d = {"k1":"olivier", "k2":"engel"}
>>> e = d.copy()
>>> d["k1"] = "XXX"
>>> e
{'k2': 'engel', 'k1': 'olivier'}
```

Les fonctions:

Une `fonction` (ou function) est une suite d'instructions que l'on peut appeler avec un nom.

Créer ma première fonction

Créons une fonction qui nous retournera un âge:

```
>>> def indique_mon_age():
...     return 30;
...
>>> indique_mon_age()
30
```

Vous ne pouvez pas copier coller ce code, vous devez entrer chaque ligne à la main et appuyer sur entrée pour retourner à la ligne. Les 3 chevrons et les 3 points sont affichés par l'interpréteur python.

Tout d'abord pour indiquer à l'interpréteur que vous voulez créer une fonction, on utiliser le mot clé `def` suivi d'un nom puis de parenthèses et ensuite d'un double point.

On remarque également qu'il y a un espace entre les 3 points et le mot clé "return", il s'agit d'une indentation, c'est à dire un espace qui améliore non seulement la lecture de la fonction mais qui indique que nous sommes toujours dans la fonction. Lorsque l'action demandée n'est plus dans la fonction, il ne faut plus indenter le texte. Pour indenter du texte, vous devez appuyer sur la `touche TAB` de votre clavier -ou dans d'autres cas créer 4 espaces manuellement.

Les paramètres

Créons une autre fonction:

```
>>> def augmente_moi(a):
...     return augmente_moi + 2
...
>>> augmente_moi(1)
3
```

Cette fonction incrémente de 2 une valeur que l'on passe en paramètre.

Il est d'ailleurs possible d'utiliser plusieurs paramètres:

```
>>> def augmente_moi(a, b):
...     return 30 + a + b
...
>>> augmente_moi(1, 2)
33
```

Si vous avez compris les principes des fonctions, vous avez compris 80% de ce qu'est la programmation.

Un paramètre est obligatoire

Lorsque vous indiquez des paramètres à une fonction, ces dernièrs doivent impérativement être renseignés sans quoi une erreur apparaitra.

```
>>> def augmente_moi(a, b):
...     return 30 + a + b
...
>>> augmente_moi(1)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: augmente_moi() takes exactly 2 arguments (1 given)
```

L'opérateur splat

L'opérateur splat : `*` est très souvent utilisé en python.

```
def ma_function(*var)
def ma_function(**var)
ma_function(*var)
ma_function(**var)
```

Une liste en paramètre

On peut récupérer les valeurs renseignées via une liste:

```
>>> def augmente_moi(*param):
...     return param[0] + param[1] + param[2]
...
>>> augmente_moi(1, 2, 3)
6
>>> augmente_moi(10, 20, 30)
60
```

Rendre obligatoire uniquement certains paramètres avec une liste

Si vous désirez rendre obligatoire uniquement certains paramètres, vous pouvez utiliser la syntaxe suivante:

```
>>> def ma_fiche(prenom, nom, *reste):
...     return prenom + " " + nom
...
>>> ma_fiche("olivier","engel")
'olivier engel'
```

On remarque que le paramètres "reste" est précédé d'une étoile `*`.

Utiliser un dictionnaire pour les paramètres

Vous pouvez utiliser un dictionnaire en paramètres pour cela vous devez ajouter une double étoile: `**`

```
>>> def ma_fiche(**parametres):
...     return parametres["prenom"]
...
>>> ma_fiche(prenom="olivier")
'olivier'
```

Utilisation de splat liste au niveau des appels de fonctions

Reprennons l'exemple de la fonction augmente_moi:

```
>>> def augmente_moi(*param):
...     return param[0] + param[1] + param[2]
...
```

Nous avons vu qu'il était possible de faire ceci:

```
>>> augmente_moi(1, 2, 3)
6
```

L'utilisation de l'étoile permet de passer par une liste:

```
>>> data = [1, 2, 3]
>>> augmente_moi(*data)
6
```

Utilisation de splat dictionnaire au niveau des appels de fonctions

Prénons l'exemple de cette fonction:

```
>>> def test(firstname="", lastname=""):
...     return "{} {}" .format(firstname, lastname)
Créons notre dictionnaire:

>>> data = {'firstname':'olivier', 'lastname':'engel'}
```

Et envoyons notre variable avec une étoile `*`

```
>>> test(*data)
'lastname firstname'
```

Puis avec deux étoiles `**`

```
>>> test(**data)
'olivier engel'
```

Portée des variables (variable globale et variable locale)

Une variable déclarée à la racine d'un module est visible dans tout ce module. On parle alors de variable globale.

```
>>> x = "hello"
>>> def test():
...     print x
...
>>> test()
hello
```

Et une variable déclarée dans une fonction ne sera visible que dans cette fonction. On parle alors de variable locale.

```
>>> x = False
>>> def test():
...     x = "hello"
...
>>> test()
>>> x
False
```

Procédure et fonctions

Pour votre culture informatique sachez qu'une fonction n'est pas obligée de renvoyer une valeur, on parlera alors dans ce cas plutôt de procédure.

Exercices

Petite mise en bouche :

Créer un ficher `t.py`

Exercie 1:

- Ecrire un programme qui affiche :

  `"Coucou les simploniens"`

- Ecrire un programme qui permet de saisir le nom de l'utilisateur et de renvoyer `"Bonjour"`, suivi de ce nom.

- Écrire un programme qui demande à l'utilisateur la saisie de a et b et affiche la somme de a et de b.
- Écrire un programme qui affiche une suite de 12 nombres dont chaque terme soit égal au triple du terme précédent.
- Écrire un programme qui affiche la suite de symboles suivante :

```
*
**
***
****
*****
******
*******
```

Exercice 2:

Conseil : utilisez l'interpréteur Python.

1. Constituez une liste semaine contenant les 7 jours de la semaine. À partir de cette liste, comment récupérez-vous seulement les 5 premiers jours de la semaine d'une part, et ceux du week-end d'autre part (utilisez pour cela l'indiçage) ? Cherchez un autre moyen pour arriver au même résultat (en utilisant un autre indiçage).
2. Trouvez deux manières pour accéder au dernier jour de la semaine.
3. Inversez les jours de la semaine en une commande.
4. Créez 4 listes hiver, printemps, ete et automne contenant les mois correspondant à ces saisons. Créez ensuite une liste saisons contenant les sous-listes hiver, printemps,ete et automne. Prévoyez ce que valent les variables suivantes, puis vérifiez-le dans l'interpréteur :
   - saisons[2]
   - saisons[1][0]
   - saisons[1:2]
   - saisons[:][1]

Comment expliquez-vous ce dernier résultat ?