

workshop



Elitech Hackathon: Python Programming Contest

Anis Ben Ammar – Elitech Education

9-10 of May 2020

Organisation pédagogique

- **Workshop :**
 - Ateliers collaboratifs
 - Durée : 1 journée
 - Participation active des candidats
- **Alterner entre :**
 - Aspects théoriques
 - Pratiques
- **Public : connaissances ?**



● Faible	47
● Moyen	39
● Bon	27
● Très Bon	3

Programmation



Python



Plan

- Python ?
- Les éléments du langage
- Les structures de contrôle
- La structuration du code : fonctions et modules
- Les structures de données natives Python
- Les fichiers en Python

Pourquoi Python ?



Pourquoi Python ?

- **Motivations :**
 - Pour les débutants ?
 - Pour les connaisseurs en programmation ?
- Se poser plutôt des questions d'ordre
 - À quoi sert ?
 - Quelles particularités?
- Principalement **3 raisons :**
 - La simplicité
 - Généricité : Multi-plateforme et Multi-fonctions
 - Popularité

Pourquoi Python ?

- **Raison 1** : La simplicité
 - Langage hautement expressif
 - Adapté pour l'apprentissage
 - Référence dans les programmes de l'éducation nationale
- Niveau syntaxique :
 - Visuellement clair
 - Pas de contraintes
 - ; {} ()
 - Facilité de maintenance
- Le prototypage **rapide** est possible grâce à la petite taille du code

```
myString1 = 'Hello'
myString2 = ' World'
i = 3
while i > 0:
    print (i)
    i -= 1
    if i == 0:
        print (myString1 + myString2)
```

Pourquoi Python ?

- **L'indentation** : Lisibilité et facilité de reprise

```
while True:

    try:
        # attente réponse client
        reponse = self.connexion.recv(4096)
        reponse = reponse.decode(encoding='UTF-8')
    except:
        # fin du thread
        break
```

- **Typage dynamique** : Pas besoin de spécifier les types des données
 - Identifier les types en fonction des opérations à appliquer sur ces variables

Pourquoi Python

- **Raison 2** : multi Plateforme
 - Fonctionne sur tous les OS : exécuter le code partout sans ajout/modification
 - Qualifié de Langage de haut niveau car il automatise la majorité des tâches de bas niveau gérées manuellement dans des langages traditionnels tels que C et C++
 - Conçu pour optimiser la **vitesse de développement**
 - Langage interprété
 - Interprété VS. Compilé Vs. Semi-interprété



Pourquoi Python

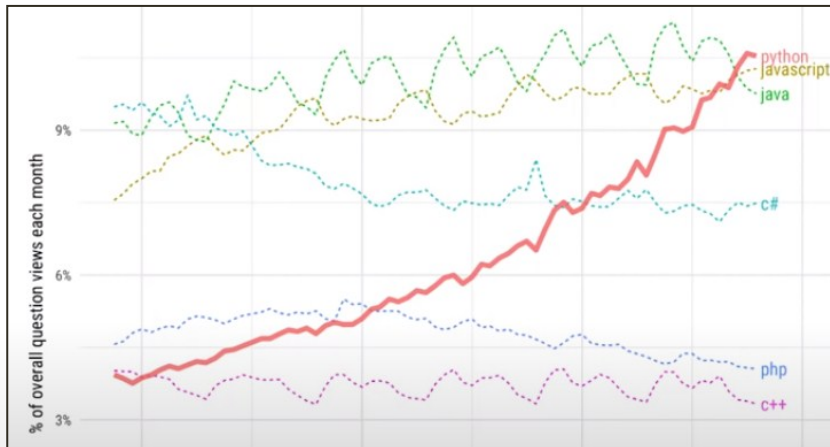
- **Multifonctions :**
 - Tous les domaines du développement web à l'embarqué
 - Cependant les utilisations majeures :
 - Développement WEB (framework Django)
 - Data Science : contexte Big Data et machine learning
 - Ecriture des scripts
 - Manipulations élémentaires / Fichiers
 - Prétraiter des données
 - Web Scrapping
 - ...



Pourquoi Python

- **3^{ème} raison** : il est populaire
 - Pas aussi récent
 - Evolution pour répondre aux nouveaux besoins aux nouvelles architectures

Evolution : stack overflow



Classement : IEEE

Language Ranking: IEEE Spectrum

Rank	Language	Type	Score
1	Python	🌐 🖥 ⚙	100.0
2	Java	🌐 📱 🖥	96.3
3	C	📱 🖥 ⚙	94.4
4	C++	📱 🖥 ⚙	87.5
5	R	🖥	81.5

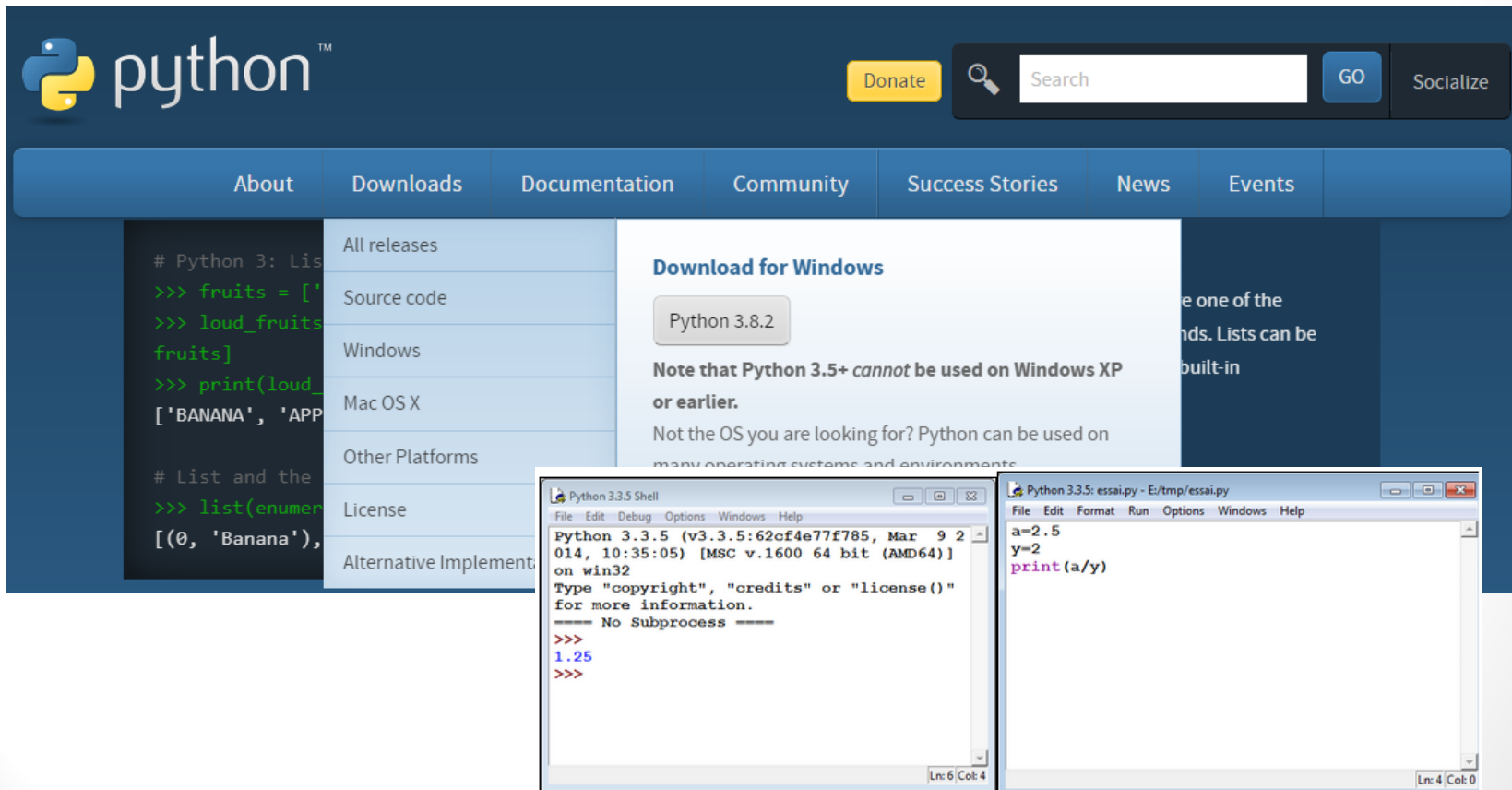
Pourquoi Python

- Communauté de développeurs python hyper active
- Ne jamais démarrer un projet à zéro
- Partage des modules sur le PyPI : Python Package Index
- Tous les géants de la technologie utilisent Python : Google, facebook, NETFLIX, Amazon, NAZA, IBM



Python : Environnement de développement

- Installation de Python 3



The screenshot displays the Python.org homepage with the Python logo and navigation links. The 'Downloads' menu is open, showing options for All releases, Source code, Windows, Mac OS X, Other Platforms, License, and Alternative Implementations. The 'Download for Windows' section is highlighted, featuring a button for Python 3.8.2 and a note that Python 3.5+ cannot be used on Windows XP or earlier. Below the website, two terminal windows are shown. The left window, titled 'Python 3.3.5 Shell', displays the output of a Python script that calculates the average of a list of fruits. The right window, titled 'Python 3.3.5: essai.py - E:/tmp/essai.py', shows the source code of the script.

```
# Python 3: Lists
>>> fruits = ['apple', 'banana', 'orange', 'pear']
>>> loud_fruits = [fruit.upper() for fruit in fruits]
>>> print(loud_fruits)
['BANANA', 'APPLE', 'ORANGE', 'PEAR']

# List and the enumerate function
>>> list(enumerate(fruits))
[(0, 'Banana'), (1, 'apple'), (2, 'orange'), (3, 'pear')]
```

Download for Windows

Python 3.8.2

Note that Python 3.5+ *cannot* be used on Windows XP or earlier.

Not the OS you are looking for? Python can be used on many operating systems and environments.

```
Python 3.3.5 Shell
Python 3.3.5 (v3.3.5:62cf4e77f785, Mar 9 2014, 10:35:05) [MSC v.1600 64 bit (AMD64)]
on win32
Type "copyright", "credits" or "license()"
>>>
1.25
>>>
```

```
Python 3.3.5: essai.py - E:/tmp/essai.py
File Edit Format Run Options Windows Help
a=2.5
y=2
print(a/y)
```







Integrated DeveLopment Environment



ANACONDA NAVIGATOR Sign in to Anaconda Cloud

Home | Environments | Projects (beta) | Learning | Community

Applications on Channels Refresh

 jupyter notebook 5.0.0 Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis. Launch	 qtconsole 4.3.0 PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more. Launch	 spyder 3.1.4 Scientific PYTHON Development Environment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features Launch	 glueviz 0.10.4 Multidimensional data visualization across files. Explore relationships within and among related datasets. Install
 orange3 3.4.1 Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox. Install	 rstudio 1.0.136 A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks. Install		

Documentation | Developer Blog | Feedback

Python : IDLE Anaconda



```
In [3]: import sys
        sys.executable

Out[3]: '/home/lqq/anaconda2/bin/python'

In [2]: from imgaug import augmenters as laa

ImportError                                Traceback (most recent call last)
<ipython-input-2-efee15aaac2> in <module>()
----> 1 from imgaug import augmenters as laa

ImportError: No module named imgaug

In [ ]:
```

Plan

- Python ?
- **Les éléments du langage**
- Les structures de contrôle
- La structuration du code : fonctions et modules
- Les structures de données natives Python
- Les fichiers en Python

Éléments du langage

- **Principaux types de données :**
 - Typage **dynamique**
 - Le type du contenu d'une variable peut donc changer si on change sa valeur
 - Le types simples
 - **int**
 - **float**
 - **bool**
 - Pas de type char mais str qui regroupe tout !
 - les types composés
 - Statiques
 - Dynamiques
 - Fonction **type()**
 - **Type None:** utilisée fréquemment pour représenter l'absence de valeur (de type NoneType)

Python : éléments du langage

- **Identifiants et mots-clés : les mots réservés**
 - Version 3.x.y : 33 mots réservés.

and	del	from	None	True
as	elif	global	nonlocal	try
assert	else	if	not	while
break	except	import	or	with
class	False	in	pass	yield
continue	finally	is	raise	
def	for	lambda	return	

Python : éléments du langage

- **Les expressions**

- Une expression est une portion de code que l'interpréteur Python peut évaluer pour obtenir une valeur.
- Les expressions peuvent être simples ou complexes. Elles sont formées d'une combinaison de littéraux, d'identifiants et d'opérateurs.

```
id1=15.3
id2=mafonction(id1)
if id2 > 0 :
    id3=math.sqrt(id2)
else:
    id4=id1-5.5*id2
```

Python : éléments du langage

- Les opérateurs arithmétiques

- +
- -
- * et **
- / , % et //

```
20 + 3 #23
20-3   #17
20*3   #60
20**3  #8000
20/3   #6.666666667
20//3  #6 Soit la division entière
20%3   #2  soit modulo ou reste de la division entière
abs (3-20) # valeur absolue|
```

Python : éléments du langage

- **Les opérateurs booléens**

- Deux valeurs possibles : False, True.
- Opérateurs de comparaison : ==, !=, >, >=, < et <=
- Opérateurs logiques : not, or et and.

```
####Les opérateurs de comparaison#####
```

```
2<8          # false
```

```
2<=8<15      #true
```

```
####Les connecteurs logiques #####
```

```
(3==3) or (9<24)    # true dès la première comparaison
```

```
(9>24) and (3==3)   #False dès la première comparaison
```

Python : éléments du langage

- Affectations : autres variantes

```
# affectation simple
v = 4

# affectation augmentée
v += 2      # idem à : v = v + 2 si v est déjà référencé

# affectation de droite à gauche
c = d = 8   # cibles multiples

# affectations parallèles d'une séquence|
e, f = 2.7, 5.1      # tuple
g, h, i = ['G', 'H', 'I'] # liste
x, y = coordonneesSouris() # retour multiple d'une fonction
```

Python : éléments du langage

- Les entrées/sorties en Python
 - Les entrées
 - fonction standard input()
 - Chaîne de caractère par défaut + formatage

```
nb_etudiant = input("Entrez le nombre d'étudiants : ")
print(type(nb_etudiant))

f1 = input("\nEntrez un flottant : ")
f1 = float(f1)

f2 = float(input("Entrez un autre flottant : "))
print(type(f2)) |
```

```
===== RESTART: C:/Python34/affic.py =====
Entrez le nombre d'étudiants : 15
<class 'str'>

Entrez un flottant : 10.5
Entrez un autre flottant : 1506
<class 'float'>
>>> |
```

Python : éléments du langage

- **Les sorties**
 - **La fonction d'affichage print**
 - tout se terminera par un saut à la ligne
 - Paramètres
 - **sep** : spécifier un séparateur
 - **end** : spécifier la marque de fin

```
a,b=2,5
print(a,b)
print("Somme : ", a+b)
print(a-b , " est la différence")
print("le produit de ", a , ' par ', b, ' est : ', a*b)
print()
print(a, end="****")
print("On a <",2**32,'> cas ', sep="##")
```

```
2 5
Somme :  7
-3  est la différence
le produit de  2  par  5  est :  10
```

```
2***On a <##4294967296##> cas
```

Python : éléments du langage

- Les sorties
 - L’affichage formaté : méthode **format()** et le **%**
 - Insérer une variable dans la chaîne à afficher
 - Pas de différence : juste une question de lisibilité et d’usage

```
print("{} {} {}".format("zéro", "un", "deux"))
print("{2} {0} {1}".format("zéro", "un", "deux"))

print("Je m'appelle {}".format("Bob"))
print("Je m'appelle {{{}}}".format("Bob"))
print("{}".format("-"*10))

a, b = 5, 3
print("The story of {c} and {d}".format(c=a+b, d=a-b))

stock = ["papier", "enveloppe", "chemise", "encre", "buvard"]
print("Nous avons de l'{} et du {} en stock\n".format(stock))
```

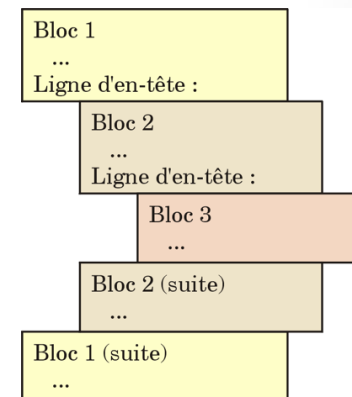
```
===== RESTART: C:/Python34/affic.py =====
zéro un deux
deux zéro un
Je m'appelle Bob
Je m'appelle {Bob}
-----
The story of 8 and 2
Nous avons de l'encre et du papier en stock
```


Plan

- Python ?
- Les éléments du langage
- **Les structures de contrôle**
- La structuration du code : fonctions et modules
- Les structures de données natives Python
- Les fichiers en Python

Python : éléments du langage

- Les contrôles des flux de séquence
 - Les instructions composées : Syntaxe
 - Une instruction composée comporte :
 - Une **ligne d'en-tête** terminée par **deux-points**
 - Un **bloc d'instructions** indenté par rapport à la ligne d'en-tête
 - Les structures
 - Conditionnelles : **if** - [**elif**] - [**else**]
 - Itératives : déterministe '**for**' et indéterministe '**While**'



Python : éléments du langage

- Structure conditionnelles : **if** - **[elif]** - **[else]**
 - Contrôler une alternative

```
if x < 0:
    print("x est négatif")

elif x % 2:
    print("x est positif et impair")

else:
    print("x n'est pas négatif et est pair")

if x: # mieux que (if x is True:) ou que (if x == True:)
    pass

|
x, y = 4, 3
# Ecriture classique :
if x < y:
    plus_petit = x
else:
    plus_petit = y
# Utilisation de l'opérateur ternaire :

plus_petit = x if x < y else y
print("Plus petit : ", plus_petit) # 3
```

← **Ecriture Compacte**

Python : éléments du langage

- Structure itératives
 - Structure indéterministe **While**

```
x, cpt = 257, 0

print("L'approximation de log2 de", x, "est", end=" ")
while x > 1:

    x //= 2 # division avec troncature
    cpt += 1 # incrémentation
print(cpt, "\n") # 8
```

```
>>>
L'approximation de log2 de 257 est 8
>>> |
```

- Contrôle de saisie

```
n = int(input("Entrez un entier [1 .. 10] : "))
while not (1 <= n <= 10):
    n = int(input("Entrez un entier [1 .. 10], S.V.P. : "))
```

```
>>>
Entrez un entier [1 .. 10] : 100
Entrez un entier [1 .. 10], S.V.P. : 120
Entrez un entier [1 .. 10], S.V.P. : 87541
Entrez un entier [1 .. 10], S.V.P. : 9
>>>
```

Python : éléments du langage

- Structure itératives
 - Structure déterministe **For**

```
for lettre in "ciao":  
    print(lettre, end=" ")  
  
for x in [2, "a", 3.14]:  
    print(x, end=" ")  
  
for i in range(5):  
    print(i, end=" ")
```

```
>>>  
c i a o 2 a 3.14 0 1 2 3 4  
>>> |
```

- Rupture de séquence : Arrêter une boucle : **break**

```
for x in range(1, 11):  
    if x == 5:  
        break  
    print(x, end=" ")  
  
print("\n Boucle interrompue pour x = ", x)
```

```
>>>  
1 2 3 4  
    Boucle interrompue pour x = 5  
>>> |
```

Python : éléments du langage

- Les instructions break et continue, et la clause else dans les boucles
- Instruction Break :
 - Arrêter l'exécution d'une boucle (while ou for)

```
y = int(input("Entrez un entier positif : "))
while not(y > 0):
    y = int(input("Entrez un entier positif, S.V.P. : "))

x = y // 2
while x > 1:
    if y % x == 0:
        print("{} a pour facteur {}".format(y, x))
        break # voici l'interruption !
        x -= 1
    else:
        print(y, "est premier.")
```

```
for x in range(1, 11):
    if x == 5:
        break
    print(x, end=" ")

print("\n Boucle interrompue pour x = ", x)
```

Python : éléments du langage

- Instruction Break :
 - Pas de structure **do... while** en Python
 - Simuler ceci à l'aide de l'instruction break

```
while True:
    n = int(input("donnez un entier positif : "))
    print("vous avez saisi ", n)
    if n > 0:
        break
print("Valeur acceptée !!! ")
```

```
donnez un entier positif : -10
vous avez saisi  -10
donnez un entier positif : 10
vous avez saisi  10
Valeur acceptée !!!
```

Python : éléments du langage

- Instruction continue
 - Passer prématurément sur la prochaine itération de la boucle

```
for i in range(4):  
    print("debut iteration", i)  
    print("bonjour")  
    if i == 2:  
        continue  
    print("fin iteration", i)  
print("apres la boucle")
```

```
debut iteration 0  
bonjour  
fin iteration 0  
debut iteration 1  
bonjour  
fin iteration 1  
debut iteration 2  
bonjour  
debut iteration 3  
bonjour  
fin iteration 3  
apres la boucle
```


Python : éléments du langage

- Clause **else**
 - définir un bloc d'instructions qui sera exécuté à la fin seulement si la boucle s'est déroulée complètement sans être interrompue par un break

```
for n in range(2, 8):  
    for x in range(2, n):  
        if n % x == 0:  
            print(n, "egale", x, "*", n/x)  
            break  
    else:  
        print(n, "est un nombre premier")
```

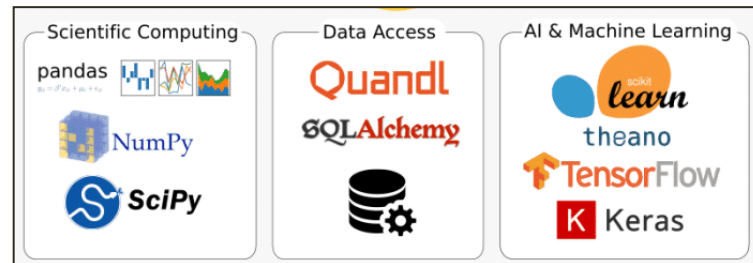
```
2 est un nombre premier  
3 est un nombre premier  
4 egale 2 * 2.0  
5 est un nombre premier  
6 egale 2 * 3.0  
7 est un nombre premier
```

Plan

- Python ?
- Les éléments du langage
- Les structures de contrôle
- **La structuration du code : fonctions et modules**
- Les structures de données natives Python
- Les fichiers en Python

Python : Structuration du code

- Mieux organiser un programme : regrouper les séquences d'instructions en :
 - fonctions / classes (**Programmation orientée objet**)
 - Modules.
 - Paquet :Package
- Fonction prédéfinies :
 - print(), input(), range()
- Packages Prédéfinis
 - Import / from ... import
 - Exemple package random



Python : Structuration du code

- **Syntaxe**

```
def nom_fonction(liste de paramètres):  
    bloc d'instructions
```

- **Typologie :**

- Fonction avec/sans arguments
- Avec/sans retour
- Fonction à nombre variable de paramètres
 - Définir des valeurs de paramètres par défaut lors de la définition d'une fonction ;
 - Utiliser une syntaxe particulière permettant de passer un nombre arbitraire d'arguments.

Python : Structuration du code

- Illustrations

```
def identity(prenom, age=18, nat="Française"):
    print('Mes données : ', prenom, " j'ai : ", age, " et je suis : ", nat )

identity("Agnes")
identity("Agnes",27)
identity("Agnes",30, "Italienne")
identity("Agnes", nat="Suisse", age=45)
```

```
def somme(*args):
    s=0
    for n in args:
        s+=n
    print(s)
somme(100,200)
somme(10,20,30)
```

```
def identity(**args):
    for i, j in args.items():
        print(i,j)

identity(prenom="Mathilde", age=25)
```

args** et *args** : les paramètres seront respectivement structurés en Tuple ou dictionnaire

Python : Structuration du code

- **Variables Globales et locales**
 - Fonction : Espace de noms → Variables **locales**
 - Espace de nom des fonctions inaccessible depuis l'extérieur
 - Programme Principal : Variables **globales**

```
def test():  
    b = 5  
    print(a, b)
```

```
a = 2  
b = 7  
test()  
print(a, b)
```

```
2 5  
2 7
```

Espace Local
b=5

Espace Global :
a=2
b=7

Python : Structuration du code

- **Usage d'une Variable Globale**
 - Besoin de définir une fonction qui soit capable de modifier une variable globale
 - **Global** : Indiquer à l'intérieur de la définition d'une fonction - quelles sont les variables à traiter globalement.

```
def test():  
    global b  
    b = 5  
    print(a, b)
```

```
a = 2  
b = 7  
test()  
print(a, b)
```

```
2 5  
2 5
```

Python : Structuration du code

- Variable locale vs globale : Illustration

```
g = 1                                # variable globale (donc visible partout)

def acces():
    # Montrer que g est bien globale.
    print("acces: g =", g)  # accès en lecture

def masquer1():
    # Illustrer le masquage de la variable globale g.
    g = 10                  # variable locale qui masque la variable globale g
    print("masquer1: g =", g)

def masquer2():
    # Illustrer l'utilisation de la variable globale g.
    # Ce n'est odnc pas du masquage. À éviter.
    global g                # on veut explicitement utiliser la variable globale g
    g = 20                  # nécessaire à cause de l'affectation ici.
    print("masquer2: g =", g)

def erreur():
    # Si on modifie une variable globale, il faut le dire explicitement.
    print("erreur: g =", g)  # devrait être la variable globale g
    g = 30                  # g locale ? ou g globale ?
```

```
def main1():|
    print("main1: g =", g)
    masquer1()
    print("main1: g =", g)

def main2():
    print("main2: g =", g)
    masquer2()
    print("main2: g =", g)

def main():
    acces()
    main1()
    main2()
    erreur()

if __name__ == "__main__":
    main()
```

```
>>>
acces: g = 1
main1: g = 1
masquer1: g = 10
main1: g = 1
main2: g = 1
masquer2: g = 20
main2: g = 20
```


Python : Structuration du code

- Un **module**
 - Fichier qui contient un regroupement de variables, classes et fonctions.
- Un **package (paquet)** : regroupement de modules
- **Directive import**
 - La directive import permet ensuite d'utiliser le code contenu dans le fichier python.
 - Import package_name
 - From package_name import * (**pas besoin de rappeler le nom du package à chaque appel**)
- **Primitive reload**
 - Lorsqu'un fichier source est modifié et déjà chargé par une directive import, les modifications ne seront pas visibles par le code.

Organisation de ses propres packages

Plan

- Python ?
- Les éléments du langage
- Les structures de contrôle
- La structuration du code : fonctions et modules
- **Les structures de données natives Python**
- Les fichiers en Python

Python : les structures de données natives



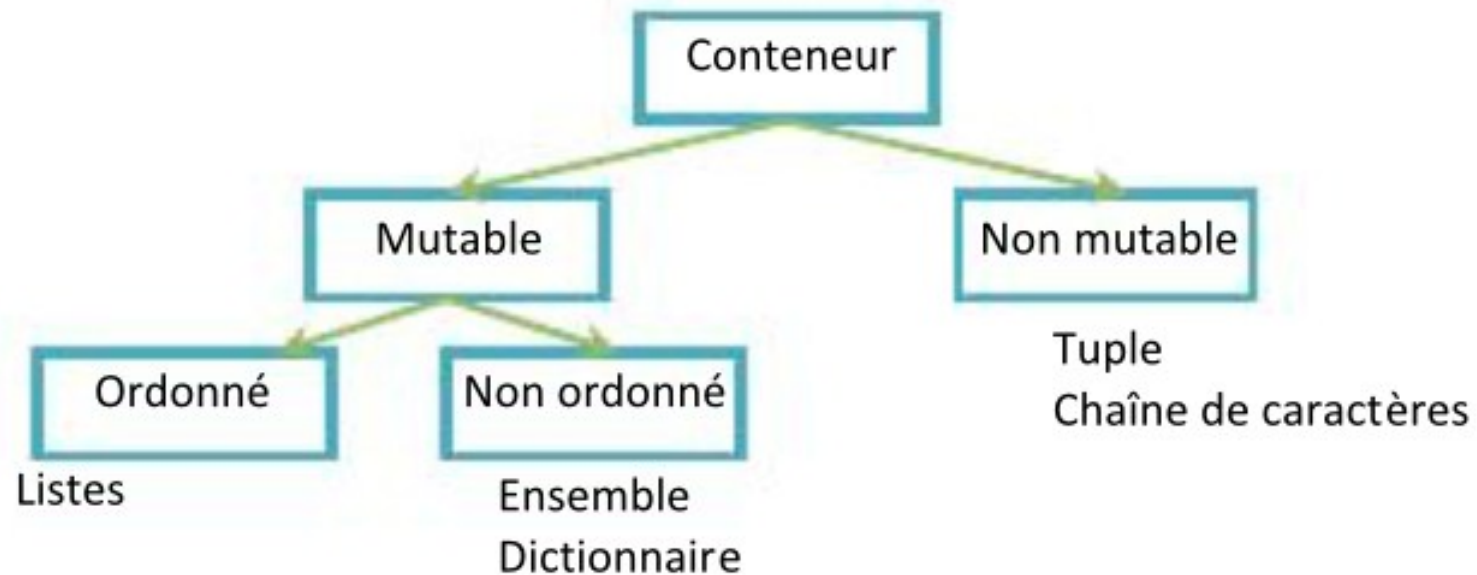
- Les conteneurs standards
 - Des moyens de stockage de données
 - **Stocker** des objets sous forme **organisée** et suivre des **règles** d'accès spécifiques
 - Ce sont des objets qui peuvent contenir toute une série d'autres objets et qui proposent des méthodes permettant de les manipuler.

Python : les structures de données natives

- **Critères** : ordonnée, doublons, mutable, itérable, indexable (accès par indice)
- **Opérations** : ajout, suppression, tri, intersection, union, différence
- **Coût/complexité** des opérations : simple ajout pourra demander une réorganisation du contenu
- **Orienter/ réfléchir** le choix des structures

Python : les structures de données natives

Taxonomie des structures de données



Python : les structures de données natives

- En Python:
 - **List**: Une collection mutable d'objets ordonnés.
 - **Set**: Une collection non ordonnée d'objets uniques et immuables
 - **Tuple**: Séquence immuable hétérogène
 - **Dictionnaire** (mémoires associatives) : Association non ordonnée de clés (keys) uniques immuables à des valeurs (values) mutables
 - **Range** (intervalle) : Une séquence arithmétique de valeurs
 - **Str**: séquence immuable homogène de caractères Unicode

Python : str

INDEX	0	1	2	3	4	5
VARIABLE	H	e	l	l	o	\0
ADRESSE	0x23451	0x23452	0x23453	0x23454	0x23455	0x23456

- Chaînes de caractères : fonctions et méthodes
 - Tester l'état d'une chaîne
 - **isupper()** et **islower()**
 - **istitle()**
 - **isalnum()**, **isalpha()**, **isdigit()** et **isspace()**
 - **Startswith** (prefix[, start[, stop]])
 - **endswith**(suffix[,start[, stop]])
 - Retourner une nouvelle chaîne
 - **lower()**, **upper()**, **capitalize()** et **swapcase()** : retournent respectivement une chaîne en minuscule, en majuscule, en minuscule commençant par une majuscule, ou en casse inversée
 - **center**(width[, fillchar]), **ljust**(width[, fillchar]) et **rjust**(width[, fillchar]) : retournent respectivement une chaîne centrée, justifiée à gauche ou à droite, complétée par le caractère **fillchar** (ou par l'espace par défaut)

Python : str

INDEX	0	1	2	3	4	5
VARIABLE	H	e	l	l	o	\0
ADRESSE	0x23451	0x23452	0x23453	0x23454	0x23455	0x23456

- **Chaînes de caractères : fonctions et méthodes**
 - Retourner une nouvelle chaîne
 - **find**(sub[, start[, stop]])
 - **rfind**() effectue le même travail en commençant par la fin.
 - **index**() et **rindex**() font de même mais produisent une erreur (exception) si la chaîne n'est pas trouvée
 - **replace**(old[, new[, count]]) : remplace count instances (toutes par défaut) de old par new
 - **split**(seps[, maxsplit]) : découpe la chaîne en maxsplit morceaux (tous par défaut).
 - **rsplit**() effectue la même chose en commençant par la fin et **striplines**() effectue ce travail avec les caractères de fin de ligne
 - **join**(seq) : concatène les chaînes du conteneur seq en intercalant la chaîne sur laquelle la méthode est appliquée
 - **Et tant d'autres méthodes et fonctions sur les chaînes de caractères !!!!!!!!!!!**

Structures de données natives

Comment vérifier si un élément existe dans la liste Python

Comment ajouter des paires clé / valeur dans un dictionnaire en Python

Vérifier si la liste contient tous les éléments d'une autre liste en Python

Vérifier si une chaîne de caractère est contenu dans une autre

Convertir une liste en tuple

Comment supprimer plusieurs élément de la liste

Comment trier un dictionnaire par clé ou par valeur

Plan

- Python ?
- Les éléments du langage
- Les structures de contrôle
- La structuration du code : fonctions et modules
- Les structures de données natives Python
- **Les fichiers en Python**

Fichiers en Python



- Fichier : Moyen de faire persister les données
- Moyens offerts par Python pour la gestion des fichiers
 - Accès
 - Lecture modification
 - Suppression
- Différents types de fichiers : texte, classeur Excel, image, vidéo, etc..
- Besoin de certains packages pour des accès formatés
 - Pandas pour les fichiers CSV
- Package OS et shutil : fournit une manière portable d'utiliser les fonctionnalités dépendantes du système d'exploitation.



Fichiers en Python



- Manipulations
 - Lectures : `read()`, `readline()` et `readlines()`
 - Écriture `write()`

```
f= open("../Data/file.txt","w")
for i in range(10):
    f.write("This is line %d\r\n" % (i+1))
f.close()
#Open the file back and read the contents
f=open("../Data/file.txt", "r")
contents =f.read()

f1 =f.readlines()
for x in f1:
    print x
```

Fichiers en Python

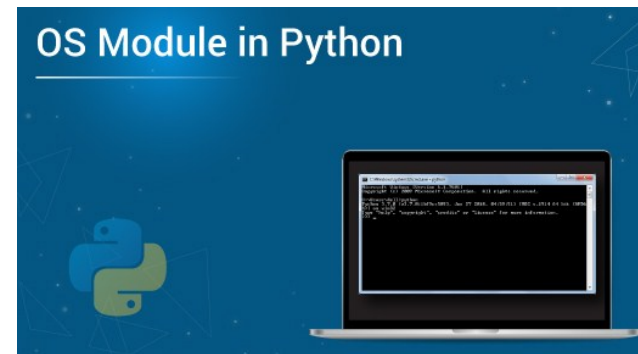


- Accès aux fichiers :
 - Open : à deux paramètres : chemin d'accès et mode d'accès
 - 'r': mode de lecture, flux positionné au début du fichier
 - 'w': Remettre la taille du fichier à nulle ou créer un fichier texte pour l'écriture. Le flux est positionné au début du fichier.
 - 'a': Ouvert pour l'écriture. Le fichier est créé s'il n'existe pas. Le flux est positionné à la fin du fichier
 - 'w+' , 'a+' et 'r+' : accès en lecture et écriture
 - Deux syntaxes possibles :
 - Files=open (path, mode)
 - With open(path, mode) as file :

Fichiers en Python



- Package OS :
 - fournit une manière portable d'utiliser les fonctionnalités dépendantes du système d'exploitation
 - Diverses directives
 - `os.chdir(path)`
 - `s.fchdir(fd)`
 - `os.getcwd()`
 - `Listdir(path)`
 - `os.path.basename(path)`
 - `os.path.exists(file_name)`
 - `Os.remove (file_name)`
 - Et bien d'autres



Fichiers en Python



- Package shutil :
 - Opérations sur les fichiers
 - Copie de fichiers
 - `shutil.copyfile`
 - Copie des métadonnées de fichiers : sans affecter les contenus
 - `shutil.copymode('myFile', 'myFile2')`
 - Manipulation d'arborescence
 - `shutil.move('myDir', 'myDir2')` : renommer
 - `shutil.copytree('myDir', 'myDir2')` : copier
 - `shutil.rmtree('myDir')` : détruire
 - Créations d'archives (.zip, .tar, etc.)
 - `shutil.make_archive('myDir', 'zip', '.', 'myDir')`

Ref : <http://www.python-simple.com/python-modules-fichiers/shutil.php>