s4-j1

October 22, 2019

1 Présentation

1.1 Objectifs

- Comprendre les principes du langage
- Savoir écrire et débugger un outil simple
- Manipuler des données et interagir avec le système
- Connaître la bibliothèque standard, éviter les pièges
- Utiliser des frameworks majeurs

1.2 Jour 1

1.2.1 Philosophie

- Principes fondamentaux
- Cas d'utilisation
- Python 2 ou Python 3 ?

1.2.2 Environnement

- Interpréteur, Documentation et IDE
- Bibliothèque standard, Linux et Windows
- Des modules pour chaque besoin

1.2.3 Notions de base

- Types, Structures, Iterable
- Fonctions, Classes, lambda et objets
- Branchements et Boucles
- Exceptions et Introspection
- Décorateurs
- Utilisation de modules

1.3 Jour 2

1.3.1 Architecture Logicielle

- Principes de base
- Les pièges à éviter

1.3.2 Interagir avec le monde réel

- Fichiers
- Journalisation
- Ligne de commande
- Processus
- Réseau

1.3.3 Solutions courantes

- Traitement, analyse de données et visualisation
- Manipulation d'images et 3D
- Interfaces graphiques
- Interaction enrichie
- Bases de données
- Applications Web

2 Philosophie

2.1 Principes fondamentaux

2.1.1 Bonnes raisons

- Lisible
- Explicite
- Simple
- Puissant
- Suffisamment rapide

2.1.2 Autres bonnes raisons

- Intuitif ! (sisi)
- API propre et stable !
- Essaye de ne proposer qu'une seule "bonne" solution pour chaque problème

2.2 Cas d'utilisation

2.2.1 Langage très largement utilisé

- Effets spéciaux au cinéma : Industrial Light & Magic (ILM)
- Contrôle aérien : Frequentis TAPtools
- Ligne de production de semiconducteurs : Philips
- Recherche de nouveaux médicaments : AstraZeneca
- Simulation de biomolécules : Molecular Modeling Toolkit…
- Plus ici ! https://www.python.org/about/success/

2.2.2 Avec des objectifs variés

- Analyse et visualisation de données
- Systèmes embarqués
- Infrastructure réseau

- Effets visuels et interfaces graphiques…

2.3 Choix d'une version

2.3.1 Pourquoi Python 3 ?

- Quelques changements de syntaxe nécessaires et incompatibles
- De nouvelles fonctionalités dans de nouvelles syntaxes
- Une gestion des exceptions plus complète
- Unicode intégré
- Les bonnes manières ont remplacé les mauvaises !

2.3.2 Python 2 vs Python 3

- Divisions

```
5 / 2 == 2   # Python 2
5 / 2 == 2.5 # Python 3
```

- Python 2 : le type *str* est utilisé pour du texte et pour une donnée binaire

```
type(bytes('toto')) == < type 'str' >
```

- En Python 2 les *str* sont des ASCII strings !! En Python 3 les *str* sont des UTF8 strings :)

```
print('En Python 3, les strings sont utf-8 \u03BCnico\u0394é!')
# En Python 3, les strings sont utf-8 nicoΔé!
```

- Python 3 : *print* est désormais une fonction !

```
print('Hello world!')
```

- Python 2 : vulnérabilité sur la méthode *input*:

```
import random
secret_number = random.randint(1,500)
print "Pick a number between 1 to 500"
while True:
    res = input("Guess the number: ")
    if res==secret_number:
        print "You win"
        break
    else:
        print "You lose"
        continue
```

2.4 Performance

2.4.1 Remarque importante

- La plupart des modules sont codés en langage bas niveau (Pyrex puis maintenant Cython)
- Les algorithmes sont éprouvés et optimisés par des experts
- L'API Python ne surcharge pas l'exécution, l'algorithme implémenté en Python est le seul coupable de la performance d'un programme ! (sauf traitement spécialisé inhabituel)

2.4.2 Bénéfice

S'il y a une perte de performance, elle est compensée par le temps de développement incroyablement réduit.

3 Environnement

3.1 Un interpréteur interactif

3.1.1 IPython3

- Programmer au fil de l'eau !
- Tester, tester et encore tester…
- Gestion complète de l'historique et des sessions de travail
- `%quickref`
- Un shell ! (âmes sensibles s'abstenir)
- Tester et écrire un script en même temps ?

```
%logstart monscript.py
x = [int(a) for a in ma_liste]
...
%logstop
```

3.1.2 Aide intégrée

- Utiliser la fonction help() à volonté
- Sans argument : mode interactif guidé
- help(x) donne la documentation de x

3.2 Consulter la documentation

3.2.1 En ligne

- https://docs.python.org/3/py-modindex.html
- https://docs.python.org/3/howto/index.html
- https://docs.python.org/3/library/index.html

3.2.2 Localement (via python3-doc)

- /usr/share/doc/python3/html/py-modindex.html
- /usr/share/doc/python3/html/howto/index.html
- /usr/share/doc/python3/html/library/index.html

3.3 Utiliser un IDE

3.3.1 Intérêt

- Contrôle de la syntaxe et de l'indentation
- Suggestion des fonctions et membres d'objets
- Gestion de projet intégré

3.3.2 Suggestions

- PyCharm
- Visual Studio Code

3.4 La bibliothèque standard

3.4.1 Thématiques couvertes

- Utilitaires, aide à l'interaction
- Support de formats et de protocoles
- Gestion de processus
- Algorithmes mathématiques
- … plus de 200 modules

3.5 Présentation de quelques modules standards 1/2

3.5.1 cmd

Construire des interfaces interactives en ligne de commande.

3.5.2 csv, json, xml...

Manipuler correctement chaque format.

3.5.3 shelve

Persistence sur disque de toute donnée "objet" en Python. Accessible via une API de type dictionnaire.

3.5.4 re

Création et utilisation d'expressions rationelles.

3.6 Présentation de quelques modules standards 2/2

3.6.1 tarfile, zipfile

Gestion macroscopique des archives et de leur contenu.

3.6.2 smtplib, imaplib, poplib, telnetlib...

Modules permettant d'implémenter des clients et serveurs.

3.6.3 multiprocessing

LE module de gestion des exécution parallélisées.

3.6.4 getpass, secrets

Modules portables d'acquisition et de génération sécurisée de secrets.

3.7 Modules externes

3.7.1 Installation

A utiliser uniquement si vous savez qu'aucun module de la bibliothèque standard ne répond à votre besoin.

Il est préférable d'installer ces modules externes via le gestionnaire de paquets de votre distribution, par exemple :

    apt-get install python3-MODULE

installera le MODULE correspondant pour Python3.

3.7.2 Version récente

Si le module n'est pas disponible, si sa version n'est pas suffisamment à jour, ou si l'on travaille dans un *virtualenv*, il est nécessaire de l'installer via :

    pip3 install MODULE

3.8 Exemples de modules externes réputés

3.8.1 python3-requests

Implémentation riche mais simple d'un client Web.

3.8.2 python3-netaddr

Gestion fine et ensembliste d'adresses IP et de sous-réseaux.

3.8.3 python3-pexpect

Instrumentalisation de commandes externes par l'attente de messages prédéfinis.

3.8.4 python3-sqlalchemy

Interaction abstraite avec plusieurs types de bases de données.

4 Environnement

4.1 Virtualenv et VirtualenvWrapper : une gestion des modules simplifiée

4.1.1 Virtualenv

- Installer :

```
pip install virtualenv
```

- Utiliser un virtualenv

```
> virtualenv monProject
> source monProject/bin/activate
(monProject) >
```

```
...
(monProject) > deactivate
```

4.1.2 VirtualenvWrapper

- Installer :

```
# Sous linux
pip install virtualenwrapper
# Sous windows
pip install virtualenvwrapper-win
```

- Gestion des virtualenvs :

```
> mkvirtualenv monProjet
> workon monProjet
(monProjet) > pip install monModule
(monProjet) > deactivate
...
> rmvirtualenv monProjet
```

5 Notions de base

5.1 Types standards

5.1.1 Entiers, flottants, chaînes de caractères, types spéciaux

[47]:
```
entier = 3
flottant = 3.141592
chaine = "string"
rien, vrai, faux = None, True, False
```

5.1.2 Expressions

[48]:
```
flottant = (flottant ** entier) - (entier ** flottant)
chaine += chaine * entier
flottant, chaine
```

[48]:
```
(-0.5380007218085119, 'stringstringstringstring')
```

5.1.3 TP

- Échanger le contenu de deux variables

5.1.4 Manipulation de chaînes

[49]:
```
phrase = "This is the voice"
phrase.swapcase().replace(" ","s").split("s")[-2:][1]
```

[49]:
```
'VOICE'
```

[50]:
```
phrase = "This %s the %s"; phrase % ("is", "voice")
```

[50]:
```
'This is the voice'
```

[51]:
```
phrase = "This %(verbe)s the %(objet)s"
phrase % {"objet": "Voice", "verbe": "is"}
```

[51]:
```
'This is the Voice'
```

5.2 Structures de données

5.2.1 Listes

[52]:
```
liste = [1, "xyz", [-0.3, [True]], "dernier"]
liste2 = [len(str(k)) for k in liste]
liste[1:3], liste[1][1], liste[-1:], liste2
```

[52]:
```
(['xyz', [-0.3, [True]]], 'y', ['dernier'], [1, 3, 14, 7])
```

[53]:
```
ma_liste = [1337, 42, 1602, 7]
sorted(ma_liste)  # Ne modifie pas ma_liste
ma_liste.sort()   # Modifie ma_liste
ma_liste
```

[53]:
```
[7, 42, 1337, 1602]
```

[54]:
```
ma_liste = [1337, 42, 1602, 7]
ma_liste[::-1]
```

[54]:
```
[7, 1602, 42, 1337]
```

5.3 Structures de données

5.3.1 Dictionnaires

[55]:
```
a = 'val'
dico = {3: "trois", a: "chaine", "liste": [42,43,44]}
dico[3], dico["liste"][1], dico['val']
```

[55]:
```
('trois', 43, 'chaine')
```

[56]:
```
user = {
    'age': 32,
    'taille': 175,
    'sexe': 'M',
    'compétences': {
```

```
        'cuisine': 3,
        'développement': 5,
        'réseau': 9
    }
}

user['compétences']['réseau']
```

[56]: 9

Attention

```
user['compétences']['Italien']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-14-6afc04fad557> in <module>()
----> 1 user['compétences']['Italien']

KeyError: 'Italien'
```

[57]:
```
user['compétences'].get('Italien', 0)
```

[57]: 0

5.4 Structures de données

5.4.1 Ensembles

[58]:
```
e1, e2 = set([1,"trois",5,7]), set([5,6,7,8])
e1.union(e2), e1.intersection(e2), e1.difference(e2)
```

[58]: ({1, 5, 6, 7, 8, 'trois'}, {5, 7}, {1, 'trois'})

5.5 Invocation de type

5.5.1 Exemple

- Chaque type peut être invoqué par sa fonction créatrice pour convertir une valeur :

```
int("123") # 123
str(-0.3) # "-0.3"
set([1,1,2,3,3,4]) # {1,2,3,4}
dict([ (1, "un"), (2, "deux"), (2, "trois") ])
# {1: "un", 2: "trois"}
```

- Certains types sont incompatibles :

```
int("trois")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-13-38e24bbb3403> in <module>()
----> 1 int("trois")

ValueError: invalid literal for int() with base 10: 'trois'
```

5.6 Branchements

5.6.1 Exemple

[59]:
```
if not e1.intersection(e2):
    print("pas d'intersection")
elif not e1.difference(e2):
    print("pas de différence")
else:
    print("autre cas")
```

```
autre cas
```

5.7 Boucles

5.7.1 Exemples

[60]:
```
for idx in range(0, 3):
    print("{}: Hello world !".format(idx))
```

```
0: Hello world !
1: Hello world !
2: Hello world !
```

[61]:
```
for element in e1:
    print("> " + str(element))
```

```
> trois
> 1
> 5
> 7
```

[62]:
```
keep_going = True
while keep_going:
    print("Pas si vite !")
    keep_going = False
```

```
Pas si vite !
```

5.7.2 Exercice

- $2^{15} = 32768$ et $3 + 2 + 7 + 6 + 8 = 26$

- Que vaut la somme des chiffres de 2^{1000} ?

5.8 Itérables

5.8.1 Définition

- Un itérable est un objet capable de retourner chacun de ses membres, un à la fois.
- **De nombreux types et objets sont intuitivement itérables !**
- Les objets Python **non** itérables sont :
  - les types
  - les fonctions
  - les booléens
  - les entiers.
- Les itérables sont :
  - les listes
  - les tuples
  - les dictionnaires (dict)
  - les ensembles (set)
  - ..mais aussi fichiers, curseurs en base de données, etc.

5.9 Itérables

5.9.1 Exemples

[63]:
```
phrase, sortie = ["this", "is", "the", "voice"], ""
for mot in phrase:
    for lettre in mot:
        sortie = lettre + sortie
sortie
```

[63]:
```
'eciovehtsisiht'
```

[64]:
```
courses = {
    'oeufs': 12,
    'riz': '500g',
    'poulet': 'Un demi',
    'oranges': '1kg'
}
for item in courses.keys():
    print('{} : {}'.format(item, courses.get(item)))
```

```
oeufs : 12
oranges : 1kg
riz : 500g
poulet : Un demi
```

[65]:
```
for key, val in courses.items():
    print('Penser à prendre : {} de {}'.format(val, key))
```

```
Penser à prendre : 12 de oeufs
Penser à prendre : 1kg de oranges
Penser à prendre : 500g de riz
Penser à prendre : Un demi de poulet
```

5.10 Fonctions

5.10.1 Définition

[66]:
```python
def f(x,y,z):
    return sum([x*(y/i) for i in z])
f(3.14, 2.123, range(5,15))
```

[66]:
```
7.787671479242981
```

[67]:
```python
def f2(x, *args, **kwargs):
    prenom = kwargs.get('prenom', '')
    nom = args[1]
    print('Bonjour {} {} !'.format(prenom, nom))

f2(3, 5, 'd\'Arc', pi=3.14, prenom='Jeanne')
```

```
Bonjour Jeanne d'Arc !
```

5.11 Portée des variables

- Expérimenter avec les portées des variables
- Tester le mot clé *static*

[68]:
```python
a = 0

def my_function():
    a = 3
    print('Inside my_function: ', a)

my_function()

print('After my_function: ', a)
```

```
Inside my_function:  3
After my_function:  0
```

5.11.1 Fonctions récursives

- Une fonction récursive est une fonction qui s'appelle elle même dans sa définition

Exemple / exercice

- Implémenter une fonction qui calcule la valeur d'un élément de la suite de Fibonacci

[69]:
```
def fibonacci(a):
    if a < 2:
        return a
    return fibonacci(a-1) + fibonacci(a-2)

fibonacci(10)
```

[69]: 55

5.11.2 Exercices

- Quelle est la somme des entiers inférieurs à 1000 qui sont multiples de 3 ou de 5 ?
- Un nombre palindrome est un entier qui se lit de la même façon à l'endroit et à l'envers. Le plus grand palindrome fait du produit de deux entiers à 2 chiffres est `9009 = 91*99`. Quel est le plus grand palindrome fait du produit de deux entiers à 3 chiffres ?

5.12 Fonctions

5.12.1 lambda

Il est possible de déclarer des fonctions qui ne contiennent qu'une expression :

```
compare = lambda personne: personne.age
etudiants = get_list_from_db("etudiants")
sorted(etudiants, key = compare)
```

```
inverse = lambda p1, p2: p2.age - p1.age
etudiants.sort(inverse)
```

5.12.2 filter() et map()

- Les fonctions lambdas sont très utiles pour utiliser ces builtins python

[70]:
```
ages = list(range(10, 40))
majeurs = filter(lambda x: x>=18, ages)
print(list(majeurs))
```

```
[18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37,
38, 39]
```

[71]:
```
def say_hello(name):
    return 'Hello {} !'.format(name)

names = ['Amanda', 'Jean', 'Alain', 'Ines']

for msg in map(say_hello, names):
    print(msg)
```

```
Hello Amanda !
Hello Jean !
```

```
Hello Alain !
Hello Ines !
```

5.12.3 Exercices

- Générer les 100 premiers carrés entiers
- Retourner les 100 premiers entiers pairs de la suite de Fibonacci
- Convertir une liste de températures en degré Celsius en degré Fahrenheit

- Générer une liste de commandes du type:

```
numéro_de_commande, nom_article, quantitée, prix
1337,'bouchon de bouteille', 150, 0.25€
```

- Exporter la liste au format csv
- Calculer la liste contenant les tuples : (numéro_de_commande, prix_commande)
  - Attention: le prix est majoré de 10€ pour toute commande < 100€ !

```
[(1337, 47.50), ... ]
```

5.13 Compréhension de liste

[72]:
```
celsius = [10.4, 20, 25.1, 17.8, 12.3]
fahrenheit = [ ((9/5)*x + 32) for x in celsius ]

for idx in range(0, len(celsius)):
    print('{} Celsius équivaut à {} Fahrenheit'.format(celsius[idx],␣
 ↪fahrenheit[idx]))
```

```
10.4 Celsius équivaut à 50.72 Fahrenheit
20 Celsius équivaut à 68.0 Fahrenheit
25.1 Celsius équivaut à 77.18 Fahrenheit
17.8 Celsius équivaut à 64.03999999999999 Fahrenheit
12.3 Celsius équivaut à 54.14 Fahrenheit
```

[73]:
```
triangles = [(x,y,z) for x in range(1,30) for y in range(x,30) for z in␣
 ↪range(y,30) if x**2 + y**2 == z**2]
triangles
```

[73]:
```
[(3, 4, 5),
 (5, 12, 13),
 (6, 8, 10),
 (7, 24, 25),
 (8, 15, 17),
 (9, 12, 15),
 (10, 24, 26),
 (12, 16, 20),
 (15, 20, 25),
 (20, 21, 29)]
```

5.13.1 Exercice

- Générer la liste des nombres premiers inférieurs à 1000

5.13.2 Remarque

- La compréhension de liste permet également de générer un type particulier d'itérables : les générateurs

[74]:
```
mon_generateur = (x*x for x in [1,2,3,4,5])
print(mon_generateur)
for carre in mon_generateur:
    print(carre)
```

```
<generator object <genexpr> at 0x7f7ba403b3b8>
1
4
9
16
25
```

5.14 Générateurs

- Un générateur est un itérable qui ne calcule l'élément suivant que lorsque qu'il est demandé
  - gain de place et de calcul !
- Pour créer un générateur, on utilise le mot clé `yield`

5.14.1 Exemple

[75]:
```
import os

def extraire_mots(dossier):
    for fichier in os.listdir(dossier):
        with open(os.path.join(dossier, fichier)) as fd:
            for ligne in fd:
                for mot in ligne.split():
                    if len(mot) > 3:
                        yield mot

generateur_mots = extraire_mots('/home/user')
for idx in range(0, 3):
    print(next(generateur_mots))
```

```
;----------------------------------------------------------------------------;
;---------------------------
Emacs
```

5.14.2 Exemple / exercice

- Réimplémenter la fonction de Fibonacci à l'aide d'un générateur

[76]:
```
def fibonacci_generateur():
    a, b = 0, 1
    while True:
        yield a
        a, b = b, a + b

fibo = fibonacci_generateur()
print([next(fibo) for idx in range(0,20)])
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584,
4181]
```

5.15 Classes

5.15.1 Définition

"Une classe est un moule, les instances de cette classe sont les pièces issues de ce moule."

5.15.2 Objets en Python

- Déclarer une *classe* revient à créer un "nouveau type d'objet".
- Les *instances* de cette classe sont les objets de ce "type".

5.16 Classes

5.16.1 Exemple

[77]:
```
class Chien:
    race = "Labrador"
    poid = None
    def pesée(self):
        return self.poid

compagnie = Chien()
compagnie.poid = 7.1
compagnie.pesée()
```

[77]:
```
7.1
```

5.17 Classes

5.17.1 Héritage de type

[78]:
```
class Animal:
    espèce = None
    poid = None
    def pesée(self):
        return "%skg" % self.poid
class Chien(Animal):
```

```
    espèce = "Canis Lupus"
    def __init__(self, p):
        self.poid = p
toby = Chien(18.7); toby.pesée()
```

[78]: '18.7kg'

5.18 Exceptions

5.18.1 Définition

Une exception est une instance d'une sous-classe de la classe "Exception" ou de cette classe elle-même.

- Cet objet particulier est instancié au moment d'une opération invalide, interromp le flux d'exécution et est communiqué depuis l'intérieur d'une fonction vers la fonction appelante.

5.19 Exceptions

5.19.1 Exemple

```
def f(x):
    return 1/x
def g(x):
    return f(x+1)
g(-1)
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 1, in g
  File "<stdin>", line 1, in f
ZeroDivisionError: division by zero
```

5.20 Exceptions

5.20.1 Exemple (fichier source)

```
$ python3 source.py
```

```
Traceback (most recent call last):
  File "source.py", line 5, in <module>
    g(-1)
  File "source.py", line 4, in g
    return f(x+1)
  File "source.py", line 2, in f
    return 1/x
ZeroDivisionError: division by zero
```

5.21 Exceptions

5.21.1 De nombreux types d'exceptions

Pour chaque tâche à réaliser, la bibliothèque standard dispose de nombreux types d'exceptions permettant de connaître les détails de l'interruption d'une fonction lors d'une opération invalide.

5.21.2 Exemples d'exceptions standard

- **KeyboardInterrupt** : lorsqu'on envoie un CTRL-C dans la console
- **KeyError** : l'appel à une clé inexistante d'un dictionnaire
- **FileNotFoundError** : tentative de lecture d'un fichier inexistant…

Il est possible de définir ses propres exceptions en déclarant une classe qui hérite d'Exception. Il faut alors "émettre" cette exception avec "raise MonException(…)".

https://docs.python.org/3/library/exceptions.html

5.22 Exceptions

5.22.1 Comment traiter les exceptions

[79]:
```
def f(x):
    return 1/x
def g(x):
    try:
        return f(x+1)
    except ZeroDivisionError:
        print("Division par zéro détectée!")
g(-1)
```

```
Division par zéro détectée!
```

5.23 Décorateurs

- Un décorateur est une fonction qui retourne une autre fonction
- Après avoir défini `mon_decorateur`, on peut décorer une fonction avec la notation `@mon_decorateur`

```
# Décorateur
def entry_exit(func):

    def wrap_func(*args, **kwargs):
        print('Avant d\'appeler ', func.__name__)
        ret = func(*args, **kwargs)
        print('Après l\'appel de ', func.__name__)
        return ret

    return wrap_func

@entry_exit
```

```
def func():
    print('Dans func')
```

5.23.1 Exercice

- Ecrire la fonction factorielle
- Ecrire un décorateur qui vérifie que f est appelé avec un entier naturel
- Ecrire un décorateur qui compte le nombre d'appels à une fonction

5.24 Introspection

5.24.1 Examiner le contenu des objets

Il est possible d'inspecter le contenu d'un objet à l'aide de la fonction "dir" :

[80]:
```
x = "test"
print(dir(x)[-5:])
print(x.title())
```

```
['swapcase', 'title', 'translate', 'upper', 'zfill']
Test
```

5.25 Introspection

5.25.1 IPython3

À l'aide de la touche de tabulation, IPython3 nous permet d'inspecter le contenu de tout objet directement :

```
x = "test"
x.<tab>
   x.capitalize   x.encode       x.format       x.isalpha
   x.casefold     x.endswith     x.format_map   x.isdecimal
   x.center       x.expandtabs   x.index        x.isdigit
```

5.25.2 Remarque

Les membres commençant par "_" ou "___" sont dits "privés" : même s'ils sont accessibles autant que les autres membres, leur utilisation directe n'est pas souhaitée, ils ne font pas partie de l'API "publique".

5.25.3 Exercice

- Déchiffrer la phrase suivante : `Zba trareny, vyf neevirag qrznva n y'nhor cne yr synap bhrfg qr yn zbagntar !`
- Implémenter un algorithme de chiffrement/déchiffrement par substitution
  - Comment générer une clée automatiquement et de façon aléatoire ? =)
- Implémenter l'algorithme de chiffrement de Vigenère

5.26 Modules

https://docs.python.org/3/py-modindex.html

L'index des modules (en ligne ou localement) est indispensable !

5.26.1 Import et utilisation

[81]:
```
import sys
sys.version
```

[81]:
```
'3.5.3 (default, Jan 19 2017, 14:11:04) \n[GCC 6.3.0 20170118]'
```

[82]:
```
import os
os.geteuid()
```

[82]:
```
1000
```

[83]:
```
import os
os.chdir('/etc/iptables')
os.listdir()
```

[83]:
```
['empty.rules', 'ip6tables.rules', 'iptables.rules']
```

5.27 Modules

5.27.1 Différentes invocations

[84]:
```
from os import path
path.realpath('/etc/alternatives/editor')
```

[84]:
```
'/usr/bin/vim.gtk'
```

[85]:
```
import os, sys
print(os); print(sys)
```

```
<module 'os' from '/home/user/venvs/notebook/lib/python3.5/os.py'>
<module 'sys' (built-in)>
```

import random
help(random)

Help on module random:

NAME random - Random variable generators.

MODULE REFERENCE https://docs.python.org/3.5/library/random

```
The following documentation is automatically generated from the Python
source files.  It may be incomplete, incorrect or include features that
```

```
are considered implementation detail and may vary between Python
implementations.  When in doubt, consult the module reference at the
location listed above.
```

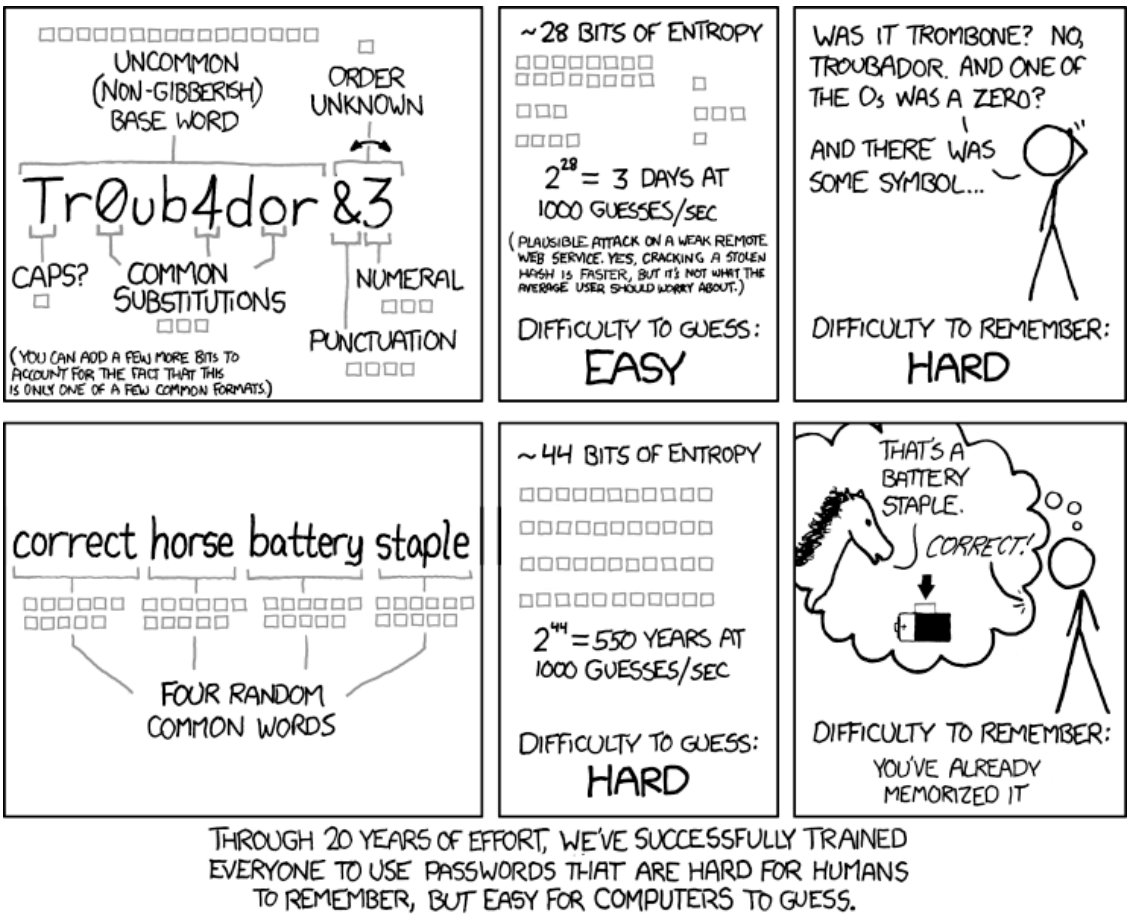
DESCRIPTION integers ——— uniform within range

```
    sequences
    ---------
           pick random element
           pick random sample
           generate random permutation
```

…

5.27.2 Exemple

Ecrire un générateur de mots de passe à l'aide du module `random` - Le module `string` pourra être utile pour forcer plusieurs cases de caractères



5.27.3 Exercice

- Implémenter un générateur de mots de passe "à la XKCD"

5.28 Modules indispensables 1/6

Rappel :

```
import MODULE
help(MODULE)
```

5.28.1 sys

- Accéder aux arguments de programme (sys.argv)
- Accès “bas niveau” à sys.stdin / sys.stdout

5.28.2 os

- Manipulations de chemins de fichiers (os.path)
- Permission des fichiers (os.chmod, os.chown, etc.)
- Attention ! avant d’utiliser os.exec* cf cours J2 :)

5.28.3 shutil

- manipulation haut-niveau des fichiers
  - copier, supprimer, déplacer, …

5.29 Modules indispensables 2/6

5.29.1 csv

- Le format CSV est plus compliqué qu’il n’y paraît !
- Les cas limites (1 sur 1M) sont ceux qui font perdre du temps en debug
- Exemple : retour à la ligne dans un champ, séparateur dans un champ, gestion de l’encodage…

5.29.2 time, date, datetime, calendar

- La gestion du temps ou des dates.
- Ces modules ont chacun un objectif différent (doc!)
- Ne pas oublier python3-pytz pour la gestion des timezones.

5.30 Modules indispensables 3/6

5.30.1 configparser

- LE module à ne pas louper pour toute utilisation de fichier de configuration !
- Fichier au format INI
- Supporte : valeurs par défaut, sections de configuration, types de données, etc.

5.30.2 argparse

- Boite à outil complète pour gérer les options en ligne de commande
- Supporte : options par défaut, longues ou courtes
- Options --usage et --help avec descriptions et paramètres typés

5.31 Modules indispensables 4/6

5.31.1 json

- Support du format JSON
- Correspondance avec un “dictionnaire” Python

5.31.2 logging

- Gestion de fichiers de log et sortie console
- Personalisation complète du format et des niveau de log
- A utiliser d’urgence

5.32 Modules indispensables 5/6

5.32.1 multiprocessing

- Une API haut niveau pour paralléliser aisément un programme
- Anticiper son utilisation pour architecturer correctement le programme
- Utiliser multiprocessing.Queue et multiprocessing.Pool

5.32.2 re

- Manipulation d’expressions rationnelles
- Utiliser x = re.compile(aiguille)
- Puis x.findall(botedefoin) (ou x.finditer(…))
- Garder la documentation sous le coude

5.33 Modules indispensables 6/6

5.33.1 shelve

- Pour tout besoin de persistence de données
- Pendant ou entre les exécutions du programme
- Permet d’organiser ses données dans un dictionnaire Python
- Celui-ci est synchronisé avec le fichier, en direct ou à la demande

5.33.2 string

- Obtenir les classes de caractères usuelles

5.34 Ipython

- `%quickref` : aide rapide

```
var = ['une', 'liste', 'comme', 'les', 'autres']
var?   # affice de l'aide sur var
```

- Exécuter des commandes :

```
!ma_cmd avec 3 arguments
var = !ma_cmd
```

- `%lsmagic`, `%magic` : aide exhaustive

- `%run` : exécute un script python
- `%edit` : éditer puis exécuter un fichier
  - `%edit 1 3 5-10` : édite avec les lignes 1 3 et 5 à 10
- `%save <fichier> 1-10` : sauvegarde les 10 premières lignes dans fichier
- `%logstart <fichier>` : permet de sauvegarder ses commandes dans fichier
  - `%logoff %logon %logstop`
- `%psearch os.g*` : recherche les commandes du module os qui commencent par 'g'
- `%who` et `%whos` : liste les variables de la session courrante
- `%hist` : retourne l'historique des commandes

[86]:
```
fichiers = !ls
print(fichiers)
```

```
['empty.rules', 'ip6tables.rules', 'iptables.rules']
```

5.35 Exercice

- Créer une copie des fichiers du dossier courant en .bak
- Créer un dossier contenant plusieurs .png
  - `!locate .png`
  - `import shutil`
- Convertir tous les .jpg du fichier courant en .png
  - `import subprocess`
  - Plus simple en bash!