

Introduction à Python

Intervenant : Jean-Christophe Gay

Mail : jean-christophe.gay@dauphine.fr

Bureau : B038

Plan du cours

1. [Introduction](#)
2. [Premiers pas avec Python](#)
3. [Utilisation de Python en Quelques Lignes](#)
4. [Contrôle de Flux](#)
5. [Programmation Fonctionnelle](#)
6. [Réalisation d'un solver de Sudoku](#)
7. [Notion d'Objet](#)
8. [Sujets de Projets](#)

Introduction

Jean-Christophe Gay :

- mail: jean-christophe.gay@dauphine.fr
- Bureau: B038 (de temps en temps et surtout le matin)

Expérience :

- 5 ans de développement en laboratoire
- 5 ans de développement pour l'Université Paris-Dauphine

Qu'est-ce que Python ?

Wikipédia :

Python est un **langage de programmation** objet, multi-paradigme et multiplateformes. Il favorise la programmation **impérative structurée, fonctionnelle et orientée objet**. Il est doté d'un **typage dynamique fort**, d'une **gestion automatique de la mémoire** par ramasse-miettes et d'un système de gestion d'**exceptions** ; il est ainsi similaire à Perl, Ruby, Scheme, Smalltalk et Tcl.

Le langage Python est placé sous une **licence libre** proche de la licence BSD et fonctionne sur la plupart des plates-formes informatiques, des supercalculateurs aux ordinateurs centraux, de Windows à Unix avec notamment GNU/Linux en passant par macOS, ou encore Android, iOS, et aussi avec Java ou encore .NET. Il est conçu pour optimiser la productivité des programmeurs en offrant des outils de haut niveau et une syntaxe simple à utiliser.

Premiers pas en Python

Pourquoi Python ?

Version de Python ?

Installation de Python

Lancement de l'interpréteur

Une première opération

On quitte

Premiers pas en Python

Pourquoi Python ?

- Nous vivons dans un monde d'information
- Il y a beaucoup trop d'informations disponible à un moment donné
- Il faut un moyen de réduire l'information, particulièrement l'information financière

Premiers pas en Python

Pourquoi Python ?

- Python est gratuit ;
- Python est puissant, flexible et simple à apprendre ;
- Python est plus adapté au Big Data que R ;
- Python se compose de nombreux modules.

Premiers pas en Python

Pourquoi
Python ?

Version de
Python ?

- Il existe deux versions concurrentes de python : 2.7 et 3.6
- Les différences majeures sont :
 - print 'Hello World' fonctionne en version 2 mais pas en 3
 - les divisions entières sont différentes
 - plus sur http://sebastianraschka.com/Articles/2014_python_2_3_key_diff.html
- Pour ce cours les différences ne seront pas très importantes.

Premiers pas en python

Pourquoi
Python ?

Version de
Python ?

Installation
de Python

Procédure d'installation :

- Se rendre sur le site de Python : www.python.org
- Section "Download" puis "Windows"
- Choisir l'installateur qui convient

Premiers pas en python

Pourquoi
Python ?

Version de
Python ?

Installation
de Python

Procédure d'installation :

- Se rendre sur le site de Python : www.continuum.io
- Section "Download" puis "Windows"
- Choisir l'installer qui convient

Premiers pas en python

Lancement de l'interpréteur

Dans un "terminal" lancer la commande `python`.

Cet interpréteur peut être utilisé pour lancer des commandes python et ainsi réaliser des calculs. Nous allons essayer rapidement.

Premiers pas en python

Lancement de l'interpréteur

Une première opération

Dans l'interpréteur :

```
>>> 1 + 1  
2  
>>> a = 1  
>>> print a  
1  
>>> a + 2  
3
```

Premiers pas en python

Lancement de l'interpréteur

Une première opération

On quitte

Dans l'interpréteur :

```
>>> quit  
Use quit() or Ctrl-D (i.e. EOF) to exit  
>>> quit()
```

Une première utilisation de Python

Si nous estimons qu'un retour de 100€ est attendu au bout d'un an avec une remise annuelle de 10%. La valeur actuelle d'une rentrée future d'argent est la suivante :

$$PV = \frac{FV}{(1 + R)^n}$$

Dans cette équation PV représente la valeur présente et FV la valeur future, R est la remise et n le nombre de périodes.

Quelle est la valeur actuelle ?

```
>>> 100 / (1 + 0.1)
```

Une première utilisation de Python

Si nous estimons qu'un retour de 100€ est attendu au bout d'un an avec une remise annuelle de 10%. La valeur actuelle d'une rentrée future d'argent est la suivante :

$$PV = \frac{FV}{(1 + R)^n}$$

Dans cette équation PV représente la valeur présente et FV la valeur future, R est la remise et n le nombre de périodes.

Quelle est la valeur actuelle ?

```
>>> 100 / (1 + 0.1)
90.90909090909090
>>>
```

Une première erreure

Que se passe-t-il pour la prochaine période ?

```
>>> 100 / (1+0.1)^2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    TypeError: unsupported operand type(s) for ^: 'float' and 'int'
>>>
```

En Python il faut utiliser `**` et non `^` pour les puissances.

```
>>> 100 / (1+0.1)**2
82.64462809917354
```

Attention à la case !

```
>>> x=2
>>> X
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'X' is not defined
>>>
```

Lorsque l'on nomme des variables ou des fonctions il faut faire attention à la case que l'on utilise. En effet Python est sensible à la case.

Type des variables

En Python les variables n'ont pas de type propre. Elles ont le type de la valeur qu'elles contiennent.

```
>>> x = 2
>>> x
2
>>> # x est un entier
>>> x="aa"
>>> x
'aa'
>>> # x est une chaîne de caractères
>>>
```

Trouver de l'aide

Facile : on en demande

```
>>> help()
```

Welcome to Python 2.7! This **is** the online help utility.

If this **is** your first time using Python, you should definitely check out the tutorial on the Internet at <http://docs.python.org/2.7/tutorial/>.

Enter the name of any module, keyword, **or** topic to get help on writing Python programs **and** using Python modules. To quit this help utility **and** **return** to the interpreter, just type "**quit**".

To get a list of available modules, keywords, **or** topics, type "**modules**", "**keywords**", **or** "**topics**". Each module also comes **with** a one-line summary of what it does; to list the modules whose summaries contain a given word such **as** "**spam**", type "**modules spam**".

```
help>
```

Trouver de l'aide

```
help> keywords
```

Here **is** a list of the Python keywords. Enter any keyword to get more help.

and

as

assert

break

class

continue

def

del

elif

else

except

exec

finally

for

from

global

if

import

in

is

lambda

not

or

pass

print

raise

return

try

while

with

yield

help>

Trouver de l'aide en ligne

- Google + Stackoverflow
- Python Site
- Tutoriels divers...

Version de Python

- Avec un terminal : **python --version** python -V
- Avec Python

```
>>> import sys
>>> print sys.version
2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609]
```

Exercices

- Trouver l'aire d'un disque de diamètre 10 cm ;
- Trouver la longueur de la diagonale d'un carré de côté 1 ;
- Quelle est la circonférence d'un champs trapézoidal dont les côtés font 127 mètres chacuns ?

Solutions

- Exercice 1

```
>>> aire=3.141592653*10**2  
>>> aire  
314.1592653  
>>>
```

- Exercice 2

```
>>> diag=2**0.5  
>>> diag  
1.4142135623730951
```

- Exercice 3

```
>>> 127 + 127 + 127 + 127  
508
```

Utilisation de Python en Quelques Lignes

Jouons avec les variables

Le module mathématique

Quelques fonctions utiles

Les tuples

Utilisation de Python en Quelques Lignes

Jouons avec les variables

- On assigne des valeurs aux variables :

```
>>> a = 1
>>> aa = 'Salut'
>>> b = 2
>>> a + b
3
```

- Afficher le contenu d'une variable :

```
>>> a
1
>>> aa
'Salut'
>>> print a
1
>>> print aa
Salut
```

Utilisation de Python en Quelques Lignes

Jouons avec les variables

- Quelques erreurs :

```
>>> aaa
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'aaa' is not defined
```

```
>>> sqrt(1)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'sqrt' is not defined
```

Utilisation de Python en Quelques Lignes

Jouons avec les variables

- Du nommage des variables

```
>>> # Mauvais exemple
>>> x = 100
>>> y = 0.1
>>> z = x / ( 1 + y )
>>> print "Le résultat est", z
Le résultat est 90.9090909091
```

```
>>> # Bon exemple
>>> FV = 100
>>> R = 0.1
>>> PV = FV / ( 1 + R )
>>> print "Le résultat est", PV
Le résultat est 90.9090909091
```

Utilisation de Python en Quelques Lignes

Jouons avec les variables

Introspection

```
>>> dir
['__builtins__', '__doc__', '__name__', '__package__']
```

dir() permet d'accéder à l'ensemble des nom définis à un instant

```
>>> a = 1
>>> dir()
['__builtins__', '__doc__', '__name__', '__package__', 'a']
```

dir(a) permet d'avoir les fonctions applicable à a :

```
>>> type(a)
<type 'int'>
>>> dir(a)
['__abs__', '__add__', '__and__', '__cmp__', ...
'denominator', 'imag', 'numerator', 'real']
```

Utilisation de Python en Quelques Lignes

Jouons avec les variables

Suppression d'une variable

```
>>> dir
<built-in function dir>
>>> dir()
['__builtins__', '__doc__', '__name__', '__package__']
>>> maVariable=1
>>> dir()
['__builtins__', '__doc__', '__name__', '__package__', 'maVar'
>>> del(maVariable)
>>> dir()
['__builtins__', '__doc__', '__name__', '__package__']
```

Utilisation de Python en Quelques Lignes

Jouons avec
les variables

Le module
mathématique

- Les opérations de base

```
>>> # Addition
>>> 1 + 1
2
>>> # Soustraction
>>> 1 - 2
-1
>>> # Multiplication
>>> 2 * 2
4
>>> # Divisions
>>> 3 / 2
1
>>> 3 / 2.
1.5
>>> 3//2.
1.0
>>> int(2.5)
2
```

Utilisation de Python en Quelques Lignes

Jouons avec
les variables

Le module
mathématique

- Plus de mathématiques, le module

```
>>> dir()
['__builtins__', '__doc__', '__name__', '__package__']
>>> import math
>>> dir()
[... , 'math']
>>> dir(math)
['__doc__', '__name__', '__package__', 'acos', 'acosh',
..., 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
>>> math.pow(2,2)
4.0
```

Utilisation de Python en Quelques Lignes

Jouons avec
les variables

Le module
mathématique

- Plus de mathématiques, le module

```
>>> help(pow)
Help on built-in function pow in module __builtin__:

pow(...)
    pow(x, y[, z]) -> number
```

With two arguments, equivalent to $x**y$. With three arguments, equivalent to $(x**y) \% z$, but may be more efficient (e.g.

```
>>> pow(3,10,4)
1
>>> 3**10
59049
>>> 59049%4
1
```

Utilisation de Python en Quelques Lignes

Jouons avec
les variables

- Choisir la précision

```
>>> 3/7.  
0.42857142857142855  
>>> payement=3/7.  
>>> pay2=round(payement,4)  
>>> pay2  
0.4286
```

Le module
mathématique

- Mais attention...

```
>>> payement*pow(10,6)  
428571.4285714285  
>>> pay2*pow(10,6)  
428600.0
```

Utilisation de Python en Quelques Lignes

Jouons avec
les variables

Le module
mathématique

- e, Pi, log et exp

```
>>> math.e
2.718281828459045
>>> math.pi
3.141592653589793
>>> math.log(math.e)
1.0
>>> math.log(math.exp(2))
2.0
```

Utilisation de Python en Quelques Lignes

Jouons avec
les variables

- import math

```
>>> dir()
['__builtins__', '__doc__', '__name__', '__package__']
>>> import math
>>> dir()
[..., 'math']
```

Le module
mathématique

Une note sur
import

- from math import

```
>>> dir()
['__builtins__', '__doc__', '__name__', '__package__']
>>> from math import pow
>>> dir()
[..., 'pow']
>>> from math import *
>>> dir()
[..., 'acos', 'acosh', 'asin', 'asinh', 'atan',
..., 'sqrt', 'tan', 'tanh', 'trunc']
```

Utilisation de Python en Quelques Lignes

Jouons avec les variables

- Affichage : print

```
>>> import math
>>> print 'Pi =', math.pi
>>> print math.pi
>>> print "Je crois que Pi vaut %f, enfin je crois, \
... ou alors c'est %f ?" % (math.pi, math.e)
```

Le module mathématique

Une note sur import

- Savoir de quel type est une variable

```
>>> x = 1
>>> type(x)
>>> x = float(x)
>>> type(x)
>>> if isinstance(x, (int, float)):
...     print "X est un nombre"
```

Fonctions utiles

Utilisation de Python en Quelques Lignes

Jouons avec
les variables

- Combinaison de chaînes de caractères

```
>>> a = 'Hello'  
>>> b = 'World'  
>>> a+b  
>>> a+' '+b  
>>> print "%s %s" % (a,b)
```

Le module
mathématique

- Strip et upper

```
>>> x = ' hello '  
>>> print x.upper()  
>>> print x.capitalize()  
>>> print x.strip()  
>>> print x.strip().capitalize()
```

Une note sur
import

Fonctions
utiles

Utilisation de Python en Quelques Lignes

Les tuples

```
>>> t = ('JC', 33)
>>> print t
>>> len(t)
>>> t[0]
>>> type(t[1])
>>> type(t)
>>> print "My name is %s and my age is %d." % t
```

Exercices

- Comment trouver toutes les fonctions built-in ?
- Convertir 'Cet exercice est facile' en lettres majuscules.
- Nous avons 41 personnes dans la classe. Si nous devons faire des groupes de 3 pour des projets, combien de groupes et combien de personnes dans un groupe non entier ?
- Expliquer le résultat suivant :

```
>>> x=5.566  
>>> round(x,2)  
5.57
```

Solutions

- Exercice 4

```
>>> dir(__builtins__)
```

- Exercice 5

```
>>> s='Cet exercice est facile'.upper()  
>>> print s
```

- Exercice 6

```
>>> print 'Avec %d élèves ont peut faire %d \  
... groupes de %d personnes et il en restera %d.' % (41,41/3,3,41%3)  
Avec 41 élèves ont peut faire 13 groupes de 3 personnes et il en restera 2.
```

Solutions

- Exercice 7 C'est le fonctionnement classique de l'arrondi en mathématique :

Si $x - \text{int}(x) < 0.5$

Alors $\text{round}(x) = \text{int}(x)$

Sinon $\text{round}(x) = \text{int}(x) + 1$

Contrôle de flux

Les Booléens

Les Opérateurs Booléen

Les blocs de code

If, then, else, elif

Contrôle de flux

Les Booléens

```
>>> spam = True
>>> spam
True
>>> true
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'true' is not defined
>>> True = 2
  File "<stdin>", line 1
SyntaxError: can't assign to keyword
```

Contrôle de flux

Les Booléens

```
>>> help('keywords')
```

Here **is** a list of the Python keywords. Enter any keyword to get more help.

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

Contrôle de flux

Les Booléens

Les Opérateurs Booléen

	Opérateur	Signification
	<code>==</code>	Égal à
	<code>!=</code>	Différent de
	<code><</code>	Inférieur à
	<code>></code>	Supérieur à
	<code><=</code>	Inférieur ou égal à
	<code>>=</code>	Supérieur ou égal à

Contrôle de flux

Les Booléens

Les Opérateurs Booléen

```
>>> 42 == 42
True
>>> 42 == 99
False
>>> 2 != 3
True
>>> 2 != 2
False
```

Contrôle de flux

Les Booléens

Les Opérateurs Booléen

```
>>> 'hello' == 'hello'  
True  
>>> 'hello' == 'Hello'  
False  
>>> 'dog' != 'cat'  
True  
>>> True == True  
True  
>>> True != False  
True  
>>> 42 == 42.0  
True  
>>> 42 == '42'  
False
```

Contrôle de flux

Les Booléens

Les Opérateurs Booléen

```
>>> (4 < 5) and (5 < 6)
True
>>> (4 < 5) and (9 < 6)
False
>>> (1 == 2) or (2 == 2)
True
>>> 2 + 2 == 4 and not 2 + 2 == 5 and 2 * 2 == 2 + 2
True
```

Contrôle de flux

Les Booléens

L'indentation c'est la vie !

Les
Opérateurs
Booléen

En python les blocs de code sont délimités par l'indentation, ils commencent en général par ':' et se prolongent tant que l'indentation est présente.

Les blocs de
code

```
>>> name = 'Mary'  
>>> password = 'swordfish'  
>>> if name == 'Mary':  
...     print('Hello Mary')  
...     if password == 'swordfish':  
...         print('Access granted.')  
...     else:  
...         print('Wrong password.')
```

Contrôle de flux

Les Booléens

```
>>> if name == 'Mary':  
...     print('Hello Mary')
```

Les
Opérateurs
Booléen

```
>>> if name == "Mary":  
...     print('Hello Mary')  
... else:  
...     print('Who are you?')
```

Les blocs de
code

```
>>> name = 'Bob'  
>>> age = 5  
>>> if name == 'Alice':  
...     print('Hi, Alice.')  
... elif age < 12:  
...     print('You are not Alice, kiddo.')
```

If, then, else,
elif

Contrôle de flux

Boucles

- Les boucles While

```
>>> spam = 0
>>> while spam < 5:
...     print('Hello, world.')
...     spam = spam + 1
```

- Break

```
>>> while True:
...     print('Please type your name.')
...     name = input()
...     if name == 'your name':
...         break
>>> print('Thank you!')
```

Contrôle de flux

Boucles

- Continue

```
>>> while True:  
...     print('Who are you?')  
...     name = input()  
...     if name != 'Joe':  
...         continue  
...     print('Hello, Joe. What is the password?')  
...     password = input()  
...     if password == 'swordfish':  
...         break  
>>> print('Access granted.')
```

Contrôle de flux

Boucles

- Les boucles For

```
>>> print('My name is')
>>> for i in range(5):
...     print('Jimmy Five Times (' + str(i) + ')')
```

range(stop) -> range object

range(start, stop[, step]) -> range object

Return an object that produces a sequence of integers
from start (inclusive) to stop (exclusive) by step.

range(i, j) produces i, i+1, i+2, ..., j-1. start
defaults to 0, and stop is omitted! range(4)
produces 0, 1, 2, 3. These are exactly the valid
indices for a list of 4 elements. When step is given,
it specifies the increment (or decrement).

Exercices

- Écrire un code qui affiche 'Hello' si 1 est stocké dans la variable spam, 'Howdy' si 2 est stocké dans spam et 'Greetings' dans tous les autres cas.
- Quelle est la différence entre break et continue ?
- Quelle sont les différences entre range(10), range(0, 10) et range(0, 10, 1) ?
- Écrire un programme qui calcule la somme des 20 premiers entiers à l'aide d'une boucle for.
- Faire la même chose à l'aide d'une boucle while.
- Faire la même chose.

Solutions

- Exercice 8:

```
if spam == 1:  
    val = 'Hello'  
elif spam == 2:  
    val = 'Howdy'  
else:  
    val = 'Greetings'  
print(val)
```

- Exercice 9:

Break permet de quitter une boucle, alors que continue permet d'aller directement à la prochaine itération de la boucle.

- Exercice 10:

Absolument aucune !

Solutions

- Exercice 11:

```
somme = 0
for i in range(1, 21):
    somme += i
print(somme)
```

- Exercice 12:

```
somme = 0
i = 1
while i < 21:
    somme += i
    i += 1
print(somme)
```

- Exercice 13:

```
somme = (20*21)//2
```

Programmation Fonctionnelle

Une première fonction

Notre propre module

Un peu de mathématiques

Utilisation de notre module

Programmation Fonctionnelle

Une première fonction

Wikipédia :

En informatique, une fonction est une entité informatique qui **encapsule** une portion de code (une séquence d'instructions) effectuant un traitement spécifique **bien identifié** relativement indépendant du reste du programme, et qui peut être **réutilisé** dans le même programme, ou dans un autre. Dans ce cas, on range souvent la fonction dans une **bibliothèque** pour la rendre disponible à d'autres projets de programmation, tout en préservant l'intégrité de son implémentation.

Programmation Fonctionnelle

Une première fonction

- Définition :

```
def maFonction():
    print "Hello World"
```

- Utilisation :

```
maFonction()
```

Programmation Fonctionnelle

Une première fonction

- Définition avec des paramètres :

```
def hello(name):  
    print('Hello ' + name)
```

- Utilisation :

```
hello('Alice')  
hello('Bob')
```

Programmation Fonctionnelle

Une première fonction

- Valeurs de retour :

```
def getAnswer(answerNumber):  
    if answerNumber == 1:  
        return "C'est sûr"  
    elif answerNumber == 2:  
        return "C'est vraisemblable"  
    elif answerNumber == 3:  
        return "Oui"  
    elif answerNumber == 4:  
        return "Retentez votre chance"  
    elif answerNumber == 5:  
        return "Redemandez plus tard"  
    elif answerNumber == 6:  
        return "Concentrez-vous et redemandez"  
    elif answerNumber == 7:  
        return "Ma réponse est non"  
    elif answerNumber == 8:  
        return "Les chances ne sont pas de votre côté"  
    elif answerNumber == 9:  
        return "Peu vraisemblable"
```

Programmation Fonctionnelle

Une première fonction

- Utilisation :

```
import random
fortune = getAnswer(random.randint(1,9))
print(fortune)
```

Programmation Fonctionnelle

Une première fonction

- Passage d'arguments et print

```
print('Hello')  
print('World')
```

- les arguments nommés ('keywords arguments')

```
print('Hello', end=' ')  
print('World')  
print('Hello', 'World')  
print('cats', 'dogs', 'mice', sep=', ', end='.\\n')
```

Programmation Fonctionnelle

Une première fonction

Portée des variables

- On ne peut pas utiliser des variables locales en dehors de leur scope...

```
def spam():
    eggs = 31337
    print(str(eggs))
spam()
print(eggs)
```

Programmation Fonctionnelle

Une première fonction

Portée des variables

- On ne peut toujours pas utiliser des variables locales en dehors de leur scope...

```
def spam():
    eggs = 31337
    bacon()
    print(eggs)

def bacon():
    ham = 101
    eggs = 0

spam()
```

Programmation Fonctionnelle

Une première fonction

Portée des variables

- Par contre on peut tout à fait utiliser des variables globales dans une fonction

```
def spam():
    print(eggs)

eggs = 31337
spam()
```

Programmation Fonctionnelle

Une première fonction

Portée des variables

- A ne SURTOUT PAS refaire, mais on peut le faire...

```
def spam():
    eggs = 'spam local'
    print(eggs)

def bacon():
    eggs = 'bacon local'
    print(eggs)
    spam()
    print(eggs)

eggs = 'global'
bacon()
```

Programmation Fonctionnelle

Une première fonction

Portée des variables

- L'utilisation de 'global'

```
def spam():
    global eggs
    eggs = spam

eggs = 'global'
spam()
print(eggs)
```

Notre propre module

Une première fonction

Portée des variables

Notre propre module

- Documentation Python :

A module is a file containing Python definitions and statements. The file name is the module name with the suffix .py appended.

- En fait un module python est simplement un fichier que l'on importe. Le fichier contient simplement les fonctions utilisables.
- Exemple :

```
>>> import suites
>>> # Importe le fichier suites.py
>>> dir(suites)
['__name__', 'fib', 'fib2']
```

Notre propre module

Une première fonction

Portée des variables

Notre propre module

Créons notre module "suites"

```
# Un module avec différentes suites

def fib(n):      # affiche la série de Fibonacci de 1 à n
    a, b = 0, 1
    while b < n:
        print(b, end=' ')
        a, b = b, a+b
    print()

def fib2(n):    # renvoie la suite de Fibonacci jusqu'à n
    res = []
    a, b = 0, 1
    while b < n:
        res.append(b)
        a, b = b, a+b
    return res
```

On sauve le fichier suites.py.

Notre propre module

Une première fonction

Notre propre module

Utilisation de notre module

- Utilisation :

```
>>> import suites
>>> suites.__name__
'suites'
>>> suites.fib(1000)
1 1 2 3 5 8 13 21 34 55 144 233 377 610 987
```

Notre propre module

Une première fonction

- Rendre le module executable
- Parce que l'on veut tester notre module
- Parce que l'on peut transformer rapidement un module en exécutable

Notre propre module

```
if __name__ == "__main__":
    import sys
    # Utilisation : python suites.py <rang>
    fib(int(sys.argv[1]))
```

Utilisation de notre module

Exercices

- Créer un programme cherchant à faire deviner un chiffre. Le joueur a six tentatives.
- Ajouter au module suite la suite de Syracuse définie par :

$$u_{n+1} = \frac{u_n}{2} \text{ si } u_n \text{ est pair, } 3 * u_n + 1 \text{ sinon.}$$

Solutions

- Exercice 14:

```
# Jeu où l'on doit deviner un nombre  
  
# imports  
  
# On demande 6 fois au joueur  
    # Si le chiffre est trop bas  
    # Si le chiffre est trop haut  
    # Sinon on sort  
  
# Si l'on a trouvé le chiffre  
# Ou pas...
```

Solutions

- Exercice 14:

```
# Jeu où l'on doit deviner un nombre
import random

secret = random.randint(1,20)
print('Je penses à un nombre entre 1 et 20.')

# On demande 6 fois au joueur
for nombreChoix in range(1,7):
    print('Alors quel nombre ?')
    choix = int(input())
    if choix < secret:
        print('Votre choix est trop petit.') # Si le chiffre est trop bas
    elif choix > secret:
        print('Votre choix est trop grand.') # le chiffre est trop haut
    else:
        break # Sinon on sort

if choix == secret:
    print('Bien joué.') # Si l'on a trouvé le chiffre
else:
    print('Pas de chance...') # Ou pas...
```

Solutions

- Exercice 15:

```
def syracuse(debut, rang):
    u = debut
    print(u, end=' ')
    for n in range(debut, rang):
        if u % 2 == 0:
            u = u / 2
        else:
            u = 3 * u + 1
    print(u, end=' ')
```

Le Jeu du Pendu

Concept

- Nous allons mettre au point un jeu du pendu
- Règles :
 - Un mot est choisi par l'ordinateur
 - A chaque tour le joueur propose une lettre
 - Si la lettre existe dans le mot elle apparaît
 - Sinon le compteur augmente et le joueur se rapproche de la fin

Le Jeu du Pendu

Concept

Mise en place

```
# imports
# Choisir un mot
# tant qu'il reste un tour
    # demander une lettre
    # test de la lettre
    # test victoire
    # incrémentation
```

Le sudoku

Règles du jeu

Un sudoku classique contient neuf lignes et neuf colonnes, donc 81 cases au total. Le but du jeu est de remplir ces cases avec des chiffres allant de 1 à 9 en veillant toujours à ce qu'un même chiffre ne figure qu'une seule fois par colonne, une seule fois par ligne, et une seule fois par carré de neuf cases.

On peut déduire deux règles

- Si une case contient un chiffre, ce chiffre ne peut pas se retrouver dans les blocs contenant cette case ;
- Si une case, du fait de la règle précédente, ne peut contenir qu'une seule valeur, alors cette valeur doit être affectée à cette case.

Le sudoku

Règles du jeu

Notations

Quelques notations et conventions :

- une grille se compose de 81 **cases** ou **square** ;
- les **colonnes** (cols) sont numérotés de 1 à 9 ;
- les **lignes** (rows) sont numérotés de A à I ;
- un bloc est un carré de 9 cases tels que definit par le jeu ;
- une **unité** est un ensemble de cases qui s'impactent les unes les autres : soit une ligne, soit une colonne, soit un bloc ;
- les **pairs** d'une case sont l'ensemble des cases qui sont dans une unité commune avec la case considérée.

Le sudoku

Règles du jeu

Notations

Quelques représentations d'une grille :

A1	A2	A3		A4	A5	A6		A7	A8	A9	
B1	B2	B3		B4	B5	B6		B7	B8	B9	
C1	C2	C3		C4	C5	C6		C7	C8	C9	
-----+-----+-----											
D1	D2	D3		D4	D5	D6		D7	D8	D9	
E1	E2	E3		E4	E5	E6		E7	E8	E9	
F1	F2	F3		F4	F5	F6		F7	F8	F9	
-----+-----+-----											
G1	G2	G3		G4	G5	G6		G7	G8	G9	
H1	H2	H3		H4	H5	H6		H7	H8	H9	
I1	I2	I3		I4	I5	I6		I7	I8	I9	

Le sudoku

Règles du jeu

Notations

Quelques représentations d'une grille :

4	1	7		3	6	9		8	2	5
6	3	2		1	5	8		9	4	7
9	5	8		7	2	4		3	1	6
-----+-----+-----										
8	2	5		4	3	7		1	6	9
7	9	1		5	8	6		4	3	2
3	4	6		9	1	2		7	5	8
-----+-----+-----										
2	8	9		6	4	3		5	7	1
5	7	3		2	9	1		6	8	4
1	6	4		8	7	5		2	9	3

Le sudoku

Règles du jeu

Notations

Les Units et les Peers :

A1	A2	A3									
B1	B2	B3									
C1	C2	C3		C4	C5	C6		C7	C8	C9	
-----+-----+-----											
D2											
E2											
F2											
-----+-----+-----											
G2											
H2											
I2											

Le sudoku

Règles du jeu

Comment résoudre donc une grille de Sudoku ?

Notations

- Lecture de la grille
- Afficher une grille
- Appliquer les contraintes
 - assigner les valeurs de base
 - éliminer les possibilités restantes
 - Si une case ne peut plus contenir qu'une valeur, l'assigner
 - Itérer le processus jusqu'à ce qu'il n'y ait plus de possibilité
- Choisir une case et tenter les possibilités restantes

Algorithme

Le sudoku -- Notes de développement

Construction des squares

- mes lignes sont A, B, C, D, ...
- mes colonnes sont 1, 2, 3, ...
- Comment avoir A1, A2, ..., I8, I9 ???

```
>>> c = '1'
>>> r = 'A'
>>> [ r + c ]
[ 'A1' ]
>>> [ r+c, r+c ]
[ 'A1', 'A1' ]
```

Donc :

```
>>> l = []
>>> for r in rows:
...     for c in cols:
...         l.append(r+c)
>>> l
['A1', 'A2', 'A3', 'B1', 'B2', 'B3', 'C1', 'C2', 'C3']
```

Le sudoku -- Notes de développement

Construction des squares

- Mais il y a mieux ?

```
>>> rows = 'ABC'  
>>> cols = '123'  
>>> [ r+c for r in rows for c in cols ]  
['A1', 'A2', 'A3', 'B1', 'B2', 'B3', 'C1', 'C2', 'C3']
```

Le sudoku -- Notes de développement

Construction des squares

- Mais il y a mieux ?

```
>>> rows = 'ABC'  
>>> cols = '123'  
>>> [ r+c for r in rows for c in cols ]  
['A1', 'A2', 'A3', 'B1', 'B2', 'B3', 'C1', 'C2', 'C3']
```

- Inlining : Technique visant à écrire le maximum de code sur la même ligne

```
>>> l = '123456789'  
>>> [ v for v in l ]  
['1', '2', '3', '4', '5', '6', '7', '8', '9']  
>>> r = []  
>>> for v in l:  
...     r.append(v)  
>>> r
```

Le sudoku -- Notes de développement

Construction des squares

- Et on peut aller plus loin avec des tests :

```
>>> l = '123456789'  
>>> [ ( int(v) if int(v)%2==0 else 0 ) for v in l ]  
[0, 2, 0, 4, 0, 6, 0, 8, 0]
```

Le sudoku -- Notes de développement

Construction
des squares

Les
dictionnaires
et les values

- les dictionnaires sont les tableaux associatifs de python

```
>>> d = dict()
>>> d[ 'A' ] = 1
>>> d[ 5 ] = 2
>>> d
{5:2, 'A':1}
>>> d[ 'A' ]
1
```

- ils permettent de stocker des valeurs associées à des clés. Plus simple à retrouver.

```
>>> d = dict( (s, c) for s in [ 'A1', 'B1' ] )
>>> d
{'B1': '123456789', 'A1': '123456789'}
```

Le sudoku -- Notes de développement

Construction
des squares

Les
dictionnaires
et les values

- Astuce pour construire un dictionnaire à partir de deux listes de même taille

```
>>> l = '123456789'  
>>> c = 'ABCDEFGHI'  
>>> dict(zip(l,c))  
{'8': 'H', '6': 'F', '3': 'C', '5': 'E', '9': 'I',  
'7': 'G', '1': 'A', '4': 'D', '2': 'B'}  
>>> dict(zip(c,l))  
{'G': '7', 'F': '6', 'C': '3', 'I': '9', 'E': '5',  
'A': '1', 'H': '8', 'B': '2', 'D': '4'}
```

Le sudoku -- Notes de développement

Construction
des squares

Les
dictionnaires
et les values

Les retours
multiples et
dict.items()

- Comment parcourir l'ensemble d'un dictionnaire ???

```
>>> d = dict(zip(l, chars))
>>> d[0]
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
KeyError: 0
>>> d['A1']
'1'
>>> for s,d in d.items():
...     print('%s => %s' % (s,d))
C3 => 9
...
A2 => 2
```

Le sudoku -- Notes de développement

La fonction all()

- Soit l un tableau, renvoyer vrai si tous les éléments de l sont divisibles par 2 :

```
>>> l = [ v for v in range(0,20,2) ]
>>> ret = True
>>> for v in l:
...     if v % 2 != 0:
...         ret = False
...         break
>>> ret
True
```

- c'est un peu long non ?

Le sudoku -- Notes de développement

La fonction all()

- Examinons all()

```
>>> all([True])
True
>>> all([True, False])
False
>>> all([True, True])
True
```

- Comment générer le tableau des tests à vérifier ?

```
>>> [ v % 2 == 0 for v in l ]
[True, True, True, True, True, True, True, True]
>>> all( [ v%2 == 0 for v in l ])
True
```

Le sudoku -- Notes de développement

La fonction all()

- Les set sont des ensembles non ordonnés d'éléments uniques. Ils sont très pratiques pour éliminer des valeurs multiples

Les sets

```
>>> l = list('12345654321')
>>> l
['1', '2', '3', '4', '5', '6', '5', '4', '3', '2', '1']
>>> l = list(set(l))
>>> l
['5', '4', '2', '6', '3', '1']
>>> l.sort()
>>> l
['1', '2', '3', '4', '5', '6']
```

TPs

- Sujet 1 :

Voici un algorithme simple permettant de générer des grilles de Sudoku :

- faire un mélange aléatoire des cases de la grille
- une par une remplir les cases avec un chiffre aléatoire en respectant les possibilités pour ne pas générer une grille impossible
- Si une contradiction apparaît, recommencer
- Si 17 cases sont remplies, alors la grille est prête Implémentez cet algorithme et résolvez 1000 grilles ainsi générées. Conserver les 10 grilles les plus lentes à être résolues

TPs

- Sujet 2 :

Vous pouvez lire dans un fichier texte des ensembles de 5 lignes. La première ligne contient une question, les 4 suivantes les propositions de réponses. Réaliser un programme qui génère 10 versions différentes de cette liste de questions et de réponses. Pour chaque version différente les questions ne seront pas dans le même ordre. Pour chaque question, les réponses ne seront pas dans le même ordre non plus.

TPs

- Modalités :

- Un seul TP est à rendre.
- Les TPs doivent être réalisés par groupes de une ou deux personnes
- La date limite pour rendre votre TP est le vendredi 27/01 17h. L'heure du mail fera foi.
- Vous rendrez vos TP par mail à jean-christophe.gay@dauphine.fr
- Le sujet du mail sera : '[Python] ... '
- La pièce jointe sera un fichier zip contenant le code et les traces d'exécutions

Un peu d'Objet

Définition

- Wikipédia

un objet est un **conteneur symbolique** et **autonome** qui contient des **informations** et des **mécanismes** concernant un **sujet**, manipulés dans un programme. Le sujet est souvent quelque chose de tangible appartenant au monde réel

Un peu d'Objet

Définition

- Moi

un objet c'est un peu comme une boîte dans laquelle on va ranger des choses. On peut y ranger des valeurs (informations) ou des transformations (des mécanismes). On peut avoir un tirroir avec des grains de poivre et un moulin à poivre.

Pour construire un objet, on a besoin d'un plan. D'un plan on peut construire de nombreux objets. En informatique un plan est une **classe**, et un meuble construit avec à l'aide d'un plan est une **instance**. Les informations contenues dans un objet sont les **attributs**. Les mécanismes sont des **méthodes**.

Un peu d'Objet

Définition

Une première classe

```
class TestClass:  
    """Une classe toute simple pour tester."""  
  
    def __init__(self):  
        self.monEntier = 1  
        self.monString = 'noir'  
  
    def afficher(self):  
        print(self.monEntier)  
        print(self.monString)  
  
t = TestClass()  
t.afficher()
```

Un peu d'Objet

Définition

Une première classe

```
class TestClass:  
    """Une classe toute simple pour tester."""  
  
        class_attribute = 0 # on va en faire un compteur  
    def __init__(self):  
        self.monEntier = 1  
        self.monString = 'noir'  
        TestClass.class_attribute += 1  
  
    def afficher(self):  
        print(self.monEntier)  
        print(self.monString)  
        print(TestClass.class_attribute)  
  
t = TestClass()  
t.afficher()  
t2 = TestClass()  
t.afficher()
```

Un peu d'Objet

Définition

Une première classe

Dir

- Les attributs spéciaux

```
t = TestClass()
dir(t)
['__class__', '__delattr__', '__dict__', '__dir__',
 '__doc__', '__eq__', '__format__', '__ge__',
 '__getattribute__', '__gt__', '__hash__', '__init__',
 '__le__', '__lt__', '__module__', '__ne__', '__new__',
 '__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
 '__sizeof__', '__str__', '__subclasshook__',
 '__weakref__', 'afficher', 'att', 'monEntier',
 'monString']
```

Un peu d'Objet

Définition

- Les méthodes spéciales
 - `__repr__`
 - `__str__`
 - `__getattr__`
 - `__add__`, `__radd__`, `__iadd__`
- Et plein d'autres !!!

Dir

Un peu d'Objet

Définition

- l'Héritage ???

Une première classe

L'héritage c'est comme dans la vraie vie. Si vous héritez de quelque chose c'est quelque chose qui n'est pas défini par vous mais qui fait quand même partie de vous.

Dir

- A quoi ça sert ?

Héritage

C'est pratique pour factoriser des attributs ou des méthodes.

Un peu d'Objet

Définition

Une première classe

Dir

Héritage

```
class Personne:  
    """Une simple classe à propos d'une personne"""  
    def __init__(self, nom, prenom):  
        self.nom = nom  
        self.prenom = prenom  
    def __str__(self):  
        return self.prenom + " " + self.nom  
  
class AgentSpecial(Personne):  
    """Classe définissant un agent spécial."""  
    def __init__(self, nom, prenom, matricule):  
        Personne.__init__(self, nom, prenom)  
        self.matricule = matricule  
    def __str__(self):  
        return "Agent " + self.nom + ", matricule " + \  
            self.matricule
```

Projets

Consignes

- le projet doit être rendu par mail à jean-christophe.gay@dauphine.fr. Le sujet du mail commencera par [Python].
- le projet sera rendu au plus tard le
- le fichier principal du projet sera exécutable, c'est à dire que l'on pourra lancer votre programme par la commande : `python3 fichier.py arguments`
- le projet comportera :
 - le code source
 - les fichiers d'entrée nécessaire à l'exécution de votre programme
 - un compte rendu
- le compte rendu se composera :
 - d'une introduction reprenant le sujet
 - d'une présentation rapide de vos travaux, explication de la démarche, des algorithmes.
 - d'une présentation plus détaillées des difficultés rencontrées, avec les moyens mis en oeuvre pour les dépasser.
 - d'une conclusion sur une possibilité d'amélioration ou d'ajout de fonctionnalités.

Projet 1

Pictos

Une image se cache sous une grille. Le joueur connaît le nombre de cases à noircir pour chaque ligne et chaque colonne de la grille. Il avance dans le jeu tour par tour en noircissant ou en blanchissant une case. Le jeu se termine lorsque le dessin est correctement affiché dans la grille.

Exemple de dessin :

```
024555420
4..00.00..
7.0000000.
7.0000000.
5..00000..
3...000...
1....0....
```

Mini-Projet 2

Affichage de la *insérez ici ce que vous voulez du jour*

Vous connaissez tous un site regroupant des collections de choses, comme DansTonChat ou lapenseedujour.net. Le but est ici d'aller moissonner votre site web favoris pour sortir une citation "du jour". Chaque appel du programme donnera une nouvelle citation.

Vous ajouterez une fonctionnalité de prédiction de la météo. En vous rendant sur le site de météo france vous récupérerez les prévision pour la journée des précipitations. En cas de forte probabilité de pluie ou de neige vous ajouterez au message précédent un avertissement. Pareil en cas de fortes chaleur ou de froid intense.

Projet 3

Le jeu de la vie

Le jeu de la vie, automate cellulaire imaginé par John Horton Conway en 1970, est probablement, à l'heure actuelle, le plus connu de tous les automates cellulaires.

Le jeu se déroule sur une grille à deux dimensions, théoriquement infinie (mais de longueur et de largeur finies et plus ou moins grandes dans la pratique), dont les cases — qu'on appelle des « cellules », par analogie avec les cellules vivantes — peuvent prendre deux états distincts : « vivantes » ou « mortes ». À chaque étape, l'évolution d'une cellule est entièrement déterminée par l'état de ses huit voisines de la façon suivante :

Une cellule morte possédant exactement trois voisines vivantes devient vivante (elle naît). Une cellule vivante possédant deux ou trois voisines vivantes le reste, sinon elle meurt.

Projet 4

Imaginez des races et sous-races (par exemple les elfes, les trolls, les trolls-cyclopes, etc.) et implémentez un système pour simuler des combats entre ces personnages. Vous pouvez choisir l'univers qui vous plaira (fantastique, science-fiction, mythologie, animaux, etc.).

Les attributs (par exemple vie, force, mana, etc.) seront modifiés selon les blessures infligées, dont la gravité sera déterminée aléatoirement. À vous de définir les règles ! Tout est possible. Un personnage aura perdu le combat quand ses points de vie seront à zéro. L'idéal serait de créer des personnages de même valeur, donnant lieu à des combats équilibrés.

Organisez ensuite un tournoi où chaque personnage se battra en duel avec chacun des autres. Il y aura 10 personnages différents, pas forcément tous de race différente. Le tournoi sera donc composé de 45 combats. Si vous préférez un combat plus "global", où plus de deux personnages se battent en même temps, à vous de définir les règles.

Le programme n'a pas besoin d'être graphique (mais il peut l'être). On devra quand même pouvoir "suivre" l'évolution du combat à l'écran d'une manière ou d'une autre...

Bibliographie

- Python for Finance, Yuxin Yan, PACK, 2014
- Automate the boring stuff with Python
- Python for Finance, O'Reilly