Initiation Python

Farid Smaï

05 avril 2025

Table des matières

Ce document est disponible en différents formats : notebook (https://gitlab.com/fsmai/formation-python/-/blob/master/notebooks/index.ipynb), jupyterlite (https://fsmai.gitlab.io/formation-python/lite/), binder (https://mybinder.org/v2/gl/fsmai%2Fformation-python/HEAD?labpath=notebooks%2Findex.ipynb), web (https://fsmai.gitlab.io/formation-python), pdf (https://fsmai.gitlab.io/formation-python/download/initiation-python.pdf).

Liens utiles : dépôt git (https://gitlab.com/fsmai/formation-python), suggestions/corrections (https://gitlab.com/fsmai/formation-python/-/issues), archive du dépôt (https://gitlab.com/fsmai/formation-python/-/archive/master/formation-python-master.zip).

0.1 A propos

Ces supports de formation ne sont que des supports, ils ne sont donc pas vraiment autonomes. Si leur lecture vous est utile, tant mieux, sinon pensez à vous tourner vers les nombreuses autres ressources disponibles sur le web.

Si vous envisagez de réutiliser ce document pour vos propres besoins, pensez à citer vos sources.

Les supports sont disponibles en différents formats :

- jupyter notebook (https://gitlab.com/fsmai/formation-python/-/blob/master/notebooks/index.ipynb)
- notebooks interactifs sur jupyterlite (https://fsmai.gitlab.io/formation-python/lite/)
- notebooks interactifs sur Binder (https://mybinder.org/v2/gl/fsmai%2Fformation-python/HEAD?labpath=notebooks%2Findex.ipynb)
- web (https://fsmai.gitlab.io/formation-python)
- pdf (https://fsmai.gitlab.io/formation-python/download/initiation-python.pdf)

Liens utiles :

- dépôt git (https://gitlab.com/fsmai/formation-python)
- suggestions/corrections (https://gitlab.com/fsmai/formation-python/-/issues)
- télécharger le dépôt (https://gitlab.com/fsmai/formation-python/-/archive/master/formation-python-master.zip)

0.1.1 Contributeurs

La version initiale de ce document a été produite par Farid Smaï et Théophile Guillon.

1 Prélude

1.1 Préparation de l'environnement de travail

Pour travailler, il nous faut :
- un interpréteur Python
- un éditeur pour le code
- une console pour exécuter du code

Il existe plusieurs manières de remplir ces conditions.

La solution proposée ici est applicable sur les principaux OS et reste à la fois *simple* et *robuste* : Spyder + Miniconda.

1.1.1 Spyder

Spyder (https://www.spyder-ide.org/) est un IDE (https://en.wikipedia.org/wiki/Integrated_development_environment) facile d'accès et orienté vers un usage scientifique de Python.

Son installation sous Windows est simple : télécharger et exécuter l'installeur (https://github.com/spyder-ide/spyder/releases/latest/download/Spyder_64bit_full.exe).

1.1.2 Miniforge

Miniforge (https://github.com/conda-forge/miniforge/) est un outil qui permet de
- gérer sur une même machine plusieurs installations de Python
- installer des bibliothèques scientifiques

Pour l'installer, télécharger et exécuter l'installeur (https://conda-forge.org/download/).

1.2 Qu'est-ce que la programmation ?

1.2.1 Elements de réflexion

- Ordinateur
  - processeur, mémoire vive / de masse, périphériques
- Programme
  - Algorithme, données, langage machine
- Grammaire
- Notions courantes
  - instruction
  - variable

    - constante
    - littérale
    - type
    - pointeur
    - **structure de données**
    - fonction
    - module/lib
    - **Structure de contrôle**
- Paradigmes de programmation
    - impératif
    - **objet**
    - fonctionnel
- Exécution
    - langage machine
    - compilateur / interpréteur
- Fonctionnalités avancées
    - ramasse-miettes
    - exceptions
    - réflexivité
    - …

1.2.2 Structure de données

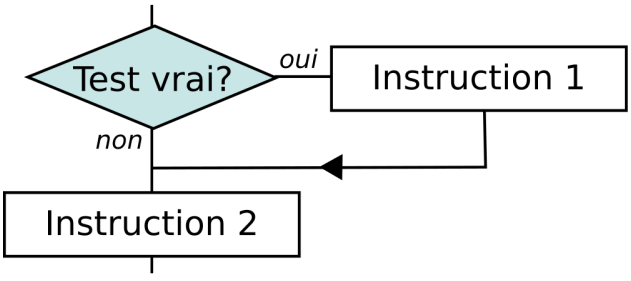
- Enregistrement
    - accès par noms fixes
    - le `struct` du C
- Tableau
    - accès par indices
    - dynamique ou statique
- Tableau associatif
    - accès par clefs
    - pas d'indice, clef -> valeur

1.2.3 Objet

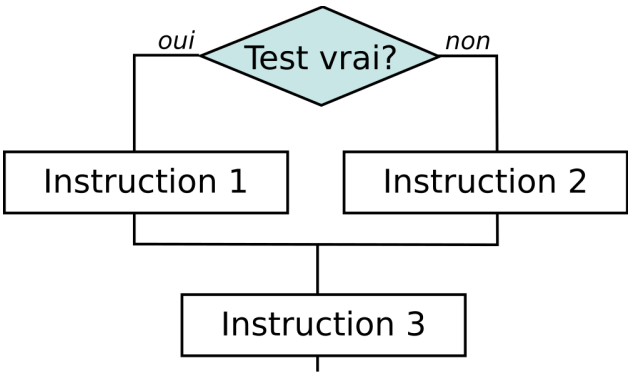
- = *Enregistrement* + méthodes associées
- Classe, instance
- Héritage

1.2.4 Structure de contrôle

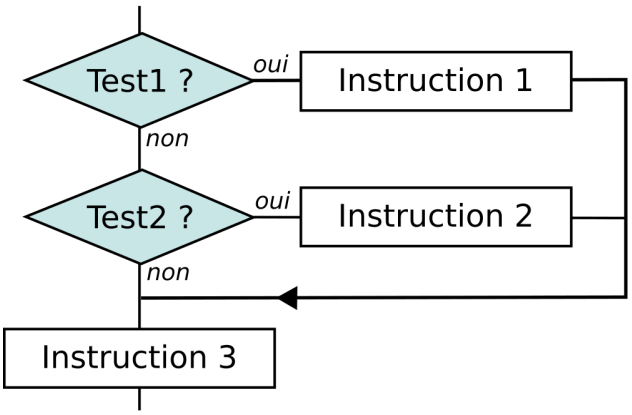
IF



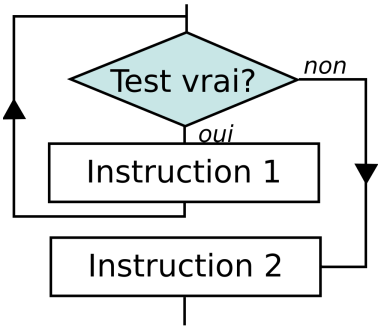
IF-ELSE



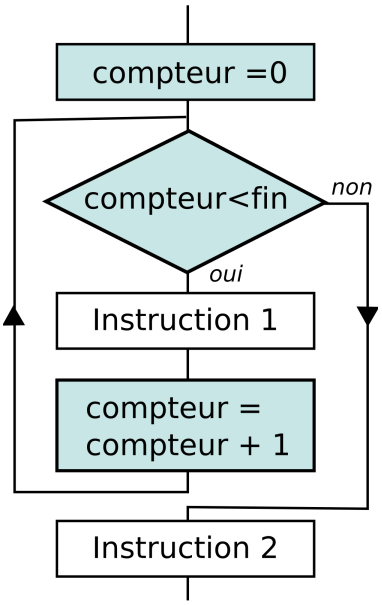
IF-ELIF



WHILE



FOR



sources :

- https://commons.wikimedia.org/wiki/File:Cf-if-fr.svg
- https://commons.wikimedia.org/wiki/File:Cf-else-fr.svg
- https://commons.wikimedia.org/wiki/File:Cf-elif-fr.svg
- https://commons.wikimedia.org/wiki/File:Cf-while-fr.svg
- https://commons.wikimedia.org/wiki/File:Cf-for-fr.svg

1.3 Python ?

1.3.1 Bref historique des langages

- dans les années 50 : FORTRAN, LISP, COBOL, ALGOL…
- dans les années 60 : PL/1, Simula, Smalltalk, Basic…
- dans les années 70 : C, PASCAL, ADA, MODULA-2…
- dans les années 80 : C++, LabView, Eiffel, Perl, VisualBasic…
- dans les années 90 : Java, tcl/Tk, Ruby, Python…
- dans les années 2000 : C#, VB.NET…

Python est un langage qui a plus de 20 ans de développement derrière lui.

1.3.2 www.python.org

Pourquoi Python ?

- gratuit, libre, open-source
- multi plateforme
- bibliothèque standard très riche
- bibliothèque tierce encore plus
- multiples usages
- particulièrement répandu dans le monde scientifique
- syntaxe claire et concise
- programmation objet / impérative
- modulaire, réutilisabilité du code

- interface avec C / C++ / Fortran
- communauté active
- documentation riche
- nombreux livres, tutoriels, cours, blog, …
- interpréteur interactif, type dynamique, garbage collector, debugger -> développement rapide
- test automatique, outils de documentation -> code robuste

Savoir coder en Python

Les bases du langage peuvent s'apprendre très rapidement.

La bibliothèque standard fournit « clef en main » un très grand nombre de fonctionnalités, à vous de vous en servir.

La maîtrise des notions avancées du langage est principalement utile dans le développement de bibliothèques (pas pour leur usage) :

- programmation impérative
  - decorateur
  - contexte
- programmation objet
  - descripteur
  - decorateur
  - meta-classe

[1]:
```
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

2 Démarrer avec Python

2.1 Les types de base

— int, float, bool, string
— utilisation des opérations arithmétiques : `+ - * / ** %`

2.1.1 Type entier

[1]:
```
# un entier
42
```

[1]:
```
42
```

[2]:
```
# addition
2 + 2
```

[2]:
```
4
```

[3]:
```
# multiplication
3 * 2
```

[3]:
```
6
```

[4]:
```
# division
5 / 2
```

[4]:
```
2.5
```

[5]:
```
# modulo
5 % 2
```

[5]:
```
1
```

[6]:
```
# division entière
5 // 2
```

[6]:
```
2
```

[7]:
```
# puissance
4 ** 3
```

[7]:
```
64
```

[8]:
```
# priorité de '*/' sur '+-'
5 * 2 - 1
```

[8]:
```
9
```

2.1.2 Type flottant

[9]:
```
# un flottant
3.14
```

[9]:
```
3.14
```

[10]:
```
# notation scientifique
5.5e3
```

[10]:
```
5500.0
```

[11]:
```
# un flottant 'à valeur entière'
5.
```

[11]:
```
5.0
```

[12]:
```
# division
5. / 2.
```

[12]:
```
2.5
```

[13]:
```
# puissance non entière
25 ** .5
```

[13]:
```
5.0
```

Conversion entier - flottant

[14]:
```
float(12)
```

[14]:
```
12.0
```

[15]:
```
int(5.0)
```

[15]:
```
5
```

[16]:
```
int(5.2)
```

[16]:
```
5
```

2.1.3 Type booléen

[17]:
```
# vrai
True
```

[17]:
```
True
```

[18]:
```
# faux
False
```

[18]:
```
False
```

Les opérateurs logiques `&` `|` sont disponibles.

Mais on préfèrera `and` `or` `not` pour cela.

[19]:
```
not False
```

[19]:
```
True
```

[20]:
```
True and False
```

[20]:
```
False
```

[21]:
```
True or False
```

[21]:
```
True
```

[22]:
```
# Attention aux priorités entre opérateurs
not True and False
```

[22]:
```
False
```

[23]:
```
# Les parenthèses sont là pour lever le doute ...
not (True and False)
```

[23]:
```
True
```

[24]:
```
# ... utilisons les !
(not True) and False
```

[24]:
```
False
```

Les booléens sont typiquement le résultat d'une comparaison

```
==    !=    >    <    <=    >=
```

[25]:
```
2 == 2
```

[25]:
```
True
```

[26]:
```
2.0 == 2
```

[26]:
```
True
```

[27]:
```
3 < 0
```

[27]:
```
False
```

[28]:
```
# on peut tester un intervalle ainsi ...
0 < 3 < 5
```

[28]:
```
True
```

[29]:
```
# ... plutôt que
(0 < 3) and (3 < 5)
```

[29]:
```
True
```

Conversion en booléen : Faux si zéro, Vrai sinon.

[30]:
```
bool(0)
```

[30]:
```
False
```

[31]:
```
bool(-3)
```

[31]:
```
True
```

[32]:
```
bool(3.14)
```

[32]:
```
True
```

Conversion d'un booléen en nombre : 0 ou 1

[33]:
```
int(True)
```

[33]:
```
1
```

[34]:
```
float(False)
```

[34]:
```
0.0
```

[35]:
```
# et donc ?
True + 2
```

[35]:
```
3
```

2.1.4 Chaines de caractères

[36]:
```
# simple quote
'bonjour'
```

[36]:
```
'bonjour'
```

[37]:
```
# double quote
"au revoir"
```

[37]:
```
'au revoir'
```

[38]:
```
# on mélange
"j'aime les pommes"
```

[38]:
```
"j'aime les pommes"
```

[39]:
```
'j\'aime les pommes'
```

[39]:
```
"j'aime les pommes"
```

[40]:
```
# saut de ligne
'''bonjour
au "revoir"
j'aime les pommes
'''
```

[40]:
```
'bonjour\nau "revoir"\nj\'aime les pommes\n'
```

[41]:
```
# oui, on peut faire plus joli
print('''bonjour
au "revoir"
j'aime les pommes
''')
```

```
bonjour
au "revoir"
j'aime les pommes
```

[42]:
```
print('bonjour\nau "revoir"\nj\'aime les pommes\n')
```

```
bonjour
au "revoir"
j'aime les pommes
```

L'arithmétique du caractère

[43]:
```
# addition
'bon' + 'jour'
```

[43]:
```
'bonjour'
```

[44]:
```
# la mutliplication
'to' * 2
```

[44]:
```
'toto'
```

[45]:
```
# dans l'autre sens
2 * 'ta'
```

[45]:
```
'tata'
```

Mais tout n'est pas permis

[46]:
```
'1' + 2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[46], line 1
----> 1 '1' + 2
```

(suite sur la page suivante)

(suite de la page précédente)

```
TypeError: can only concatenate str (not "int") to str
```

[47]:
```
'ta' * 'to'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[47], line 1
----> 1 'ta' * 'to'

TypeError: can't multiply sequence by non-int of type 'str'
```

Conversion en nombre

[48]:
```
int('-12')
```

[48]:
```
-12
```

[49]:
```
float('-1.234E+5')
```

[49]:
```
-123400.0
```

[50]:
```
int(float('3.14'))
```

[50]:
```
3
```

Conversion en booléen : Faux si “”, Vrai sinon

[51]:
```
bool('')
```

[51]:
```
False
```

[52]:
```
bool('0')
```

[52]:
```
True
```

[53]:
```
bool('False')
```

[53]:
```
True
```

Conversion en chaîne de caractère

[54]:
```
str(-12)
```

[54]:
```
'-12'
```

[55]:
```
str(3.14)
```

[55]:
```
'3.14'
```

[56]:
```
str(-1.234E+5)
```

[56]:
```
'-123400.0'
```

2.1.5 Exercice

Utiliser l'interpréteur interactif comme une calculette à int, float et string.

(*Collectionner les erreurs pour gagner des points bonus*)

2.2 *print*

Pour faire une sortie écran, on utilise `print`

[1]:
```
print(2)
print(3.14)
print('choucroute')
print(True)
print()
print('fin')
```

```
2
3.14
choucroute
True

fin
```

[2]:
```
# afficher plusieurs choses sur une même ligne
print(1, 2, 3, 4)
print(2, 3.14, 'choucroute', True)
```

```
1 2 3 4
2 3.14 choucroute True
```

[1]:
```
# plusieurs ligne de code, une seule affichée
print(2, end=' ')
print(3.14, end=' ')
print('choucroute', end=' ')
print(True, end=' ')
```

```
2 3.14 choucroute True
```

2.2.1 Exercice

Mon premier script : Ecrire *dans un fichier* des instructions `print` puis le lancer pour voir le résultat.

2.3 Exécuter son script

Il existe plein d'outils pour aider à programmer en python.

Néanmoins, on peut toujours s'en sortir avec un simple éditeur et la ligne de commande.

Dans un shell, avec un `PATH`, correctement configuré, on exécute un script en tapant :

```
> python chemin/vers/mon_script.py
```

2.4 Noms et variables

2.4.1 Utiliser des variables pour être plus flexible

La même instruction, `print(a * 2)` , donne différents résultats selon la valeur de `a`.

-> séparation entre opérations et données

[1]:
```
a = 'to'
print(a * 2)

a = 'ta'
print(a * 2)

a = 21
print(a * 2)
```

```
toto
tata
42
```

Un même nom peut être utilisé successivement pour différentes variables/valeurs sans contrainte de type.

Utiliser un nom inconnu est difficile

[2]:
```
inconnu
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[2], line 1
----> 1 inconnu

NameError: name 'inconnu' is not defined
```

[3]:
```
inconnu = 'XXX'
inconnu
```

[3]:
```
'XXX'
```

2.4.2 Qu'est-ce qu'un nom licite ?

Il utilise seulement les caractères suivants

- underscore : _
- lettres minuscules : `[a-z]`
- lettres majuscules : `[A-Z]`
- nombres : `[0-9]`

MAIS il ne commence pas par un nombre (`[0-9]`)

[4]:
```
# exemples qui marchent

x = 2

camion = 'vroum'

HAUT = 'O'
```

(suite sur la page suivante)

(suite de la page précédente)

```
unnomlongmaispaslisible = 0

un_nom_long_lisible = 1

UnNomLongAussiLisible = 2

_avec_une_patte_devant = 3

__avec_des_longues_pattes_partout__ = 4
```

[5]:
```
# exemple qui marche pas
allo? = 3
```

```
  Cell In[5], line 2
    allo? = 3
         ^
SyntaxError: invalid syntax
```

[6]:
```
# exemple qui marche pas
deux-nom = 3
```

```
  Cell In[6], line 2
    deux-nom = 3
    ^
SyntaxError: cannot assign to expression here. Maybe you meant '==' instead of '='?
```

[7]:
```
# exemple qui marche pas
0_commence_avec_zero = False
```

```
  Cell In[7], line 2
    0_commence_avec_zero = False
     ^
SyntaxError: invalid decimal literal
```

De plus, Il ne doit pas être déjà réservé par le langage

[8]:
```
from keyword import kwlist

width = 6

print(f'Voici les {len(kwlist)} mots-clefs de Python:\n')

for i in range(0, len(kwlist), width):
    for word in kwlist[i:i + width]:
        print('-', word.ljust(12), end=' ')
    print()
```

```
Voici les 35 mots-clefs de Python:
```

(suite sur la page suivante)

(suite de la page précédente)

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

[9]:
```
lambda = 123
```

```
  Cell In[9], line 1
    lambda = 123
           ^
SyntaxError: invalid syntax
```

[10]:
```
lambda_ = 123
```

2.4.3 Extra

[11]:
```
# Attention : 'int', 'float', 'str' ne sont pas réservés !
print(int(3.14))
int = 'oups'
print(int(3.14))
```

```
3
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[11], line 4
      2 print(int(3.14))
      3 int = 'oups'
----> 4 print(int(3.14))

TypeError: 'str' object is not callable
```

[12]:
```
int = type(1)  # un moyen de restaurer 'int'
```

2.4.4 Exercice

Jouer un peu avec les opérateurs `+`, `*` …

Essayer un exemple concret avec la conversion de température °C/°F *de formule $ Celsius = :nbsphinx-math :*frac{5}{9}`(Fahrenheit - 32) $ <https://fr.wikipedia.org/wiki/Degr%C3%A9_Fahrenheit>`__.

Expérimenter l'utilisation du `print(nom)` pour voir le résultat de quelques calculs.

2.5 Introduction aux fonctions

[1]:
```
def add(x, y):
    "additionne x et y"
```

(suite sur la page suivante)

(suite de la page précédente)

```
    result = x + y
    return result
```

[2]:
```
add(1, 2)
```

[2]:
```
3
```

[3]:
```
add('bon', 'jour')
```

[3]:
```
'bonjour'
```

[4]:
```
help(add)
```

```
Help on function add in module __main__:

add(x, y)
    additionne x et y
```

[5]:
```
a = add(1, 2)
b = add(3, 4)
c = add(a, b)
print(a, b, c)
```

```
3 7 10
```

[6]:
```
add(1, add(1, add(1, add(1, 1))))
```

[6]:
```
5
```

[7]:
```
from IPython.display import IFrame
IFrame("https://pythontutor.com/iframe-embed.html#code=def%20add%28x,%20y%29%3A%0A%20
↪%20%20%20%22additionne%20x%20et%20y%22%0A%20%20%20%20result%20%3D%20x%20%2B%20y%0A
↪%20%20%20%20return%20result%0A%0Aa%20%3D%20add%281,%202%29%0Ab%20%3D%20add%28a,%2010
↪%29%0A&codeDivHeight=400&codeDivWidth=350&cumulative=true&curInstr=0&
↪heapPrimitives=false&origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%5D&
↪textReferences=false",
      width='100%', height=400)
```

[7]:
```
<IPython.lib.display.IFrame at 0x222e31cbe50>
```

lien (http://pythontutor.com/visualize.html#code=def%20add%28x,%20y%29%3A%0A%20%20%20%20%22additionne%20x%20et%20y%22%0A%20%20%20%20result%20%3D%20x%20%2B%20y%0A%20%20%20%20return%20result%0A%0Aa%20%3D%20add%281,%202%29%0Ab%20%3D%20add%28a,%2010%29%0A&cumulative=true&curInstr=0&heapPrimitives=false&origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false)

2.5.1 Fonctions sans retour

[8]:
```
def ne_renvoie_rien(arg):
    "affiche arg"
    print(arg)
```

(suite sur la page suivante)

(suite de la page précédente)

```
a = ne_renvoie_rien('bonjour')
b = ne_renvoie_rien(42)
print(a, b)
```

```
bonjour
42
None None
```

[9]:
```
def ne_renvoie_rien_bis(arg):
    "affiche arg"
    print(arg)
    return

a = ne_renvoie_rien_bis('bonjour')
b = ne_renvoie_rien_bis(42)
print(a, b)
```

```
bonjour
42
None None
```

2.5.2 None ?

— `None` sert à représenter le rien (ce qui ne sert pas à rien)
— Il est unique
— A part sa conversion en booléen (`False`), il ne fonctionne avec aucun opérateur
— On teste si une variable est None avec `x is None`

[10]:
```
bool(None)
```

[10]:
```
False
```

[11]:
```
int(None)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[11], line 1
----> 1 int(None)

TypeError: int() argument must be a string, a bytes-like object or a real number, not
↪'NoneType'
```

[12]:
```
None + 0
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[12], line 1
----> 1 None + 0

TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'
```

2.5.3 Exercices

— *Fonction double*
— *Conversion de température*
— *Conversion de secondes (étape 1)*

2.6 *type()* et *help()* : introspection

— En python, *tout* est objet.
— Le type des objets est fixe.
— Un même nom peut désigner tour à tour des objets de types différents : pas de « déclaration des variables ».

Comment s'y retrouver ?

2.6.1 Qui es-tu ?

[1]:
```
# type() nous indique le type d'un objet
print(type(2))
print(type(3.14))
print(type('toto'))
```

```
<class 'int'>
<class 'float'>
<class 'str'>
```

[2]:
```
type(2) == int
```

[2]:
```
True
```

[3]:
```
# Quel est le type de 'int' ?
print(type(int))
```

```
<class 'type'>
```

[4]:
```
# Quel est le type de 'type' ?
print(type(type(int)))
```

```
<class 'type'>
```

2.6.2 Que fais-tu ?

Plus que le type d'un objet, on s'intéressera à ce qu'il sait faire.

[5]:
```
help(2)
```

```
Help on int object:

class int(object)
 |  int([x]) -> integer
 |  int(x, base=10) -> integer
 |
 |  Convert a number or string to an integer, or return 0 if no arguments
 |  are given.  If x is a number, return x.__int__().  For floating point
 |  numbers, this truncates towards zero.
```

(suite sur la page suivante)

(suite de la page précédente)

```
|
|  If x is not a number or if base is given, then x must be a string,
|  bytes, or bytearray instance representing an integer literal in the
|  given base.  The literal can be preceded by '+' or '-' and be surrounded
|  by whitespace.  The base defaults to 10.  Valid bases are 0 and 2-36.
|  Base 0 means to interpret the base from the string as an integer literal.
|  >>> int('0b100', base=0)
|  4
|
|  Built-in subclasses:
|      bool
|
|  Methods defined here:
|
|  __abs__(self, /)
|      abs(self)
|
|  __add__(self, value, /)
|      Return self+value.
|
|  __and__(self, value, /)
|      Return self&value.
|
|  __bool__(self, /)
|      True if self else False
|
|  __ceil__(…)
|      Ceiling of an Integral returns itself.
|
|  __divmod__(self, value, /)
|      Return divmod(self, value).
|
|  __eq__(self, value, /)
|      Return self==value.
|
|  __float__(self, /)
|      float(self)
|
|  __floor__(…)
|      Flooring an Integral returns itself.
|
|  __floordiv__(self, value, /)
|      Return self//value.
|
|  __format__(self, format_spec, /)
|      Default object formatter.
|
|  __ge__(self, value, /)
|      Return self>=value.
|
|  __getattribute__(self, name, /)
|      Return getattr(self, name).
```

(suite sur la page suivante)

(suite de la page précédente)

```
 |
 |  __getnewargs__(self, /)
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __hash__(self, /)
 |      Return hash(self).
 |
 |  __index__(self, /)
 |      Return self converted to an integer, if self is suitable for use as an index␣
↪into a list.
 |
 |  __int__(self, /)
 |      int(self)
 |
 |  __invert__(self, /)
 |      ~self
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __lshift__(self, value, /)
 |      Return self<<value.
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __mod__(self, value, /)
 |      Return self%value.
 |
 |  __mul__(self, value, /)
 |      Return self*value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __neg__(self, /)
 |      -self
 |
 |  __or__(self, value, /)
 |      Return self|value.
 |
 |  __pos__(self, /)
 |      +self
 |
 |  __pow__(self, value, mod=None, /)
 |      Return pow(self, value, mod).
 |
 |  __radd__(self, value, /)
 |      Return value+self.
 |
```

(suite sur la page suivante)

(suite de la page précédente)

```
|  __rand__(self, value, /)
|      Return value&self.
|
|  __rdivmod__(self, value, /)
|      Return divmod(value, self).
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __rfloordiv__(self, value, /)
|      Return value//self.
|
|  __rlshift__(self, value, /)
|      Return value<<self.
|
|  __rmod__(self, value, /)
|      Return value%self.
|
|  __rmul__(self, value, /)
|      Return value*self.
|
|  __ror__(self, value, /)
|      Return value|self.
|
|  __round__(…)
|      Rounding an Integral returns itself.
|
|      Rounding with an ndigits argument also returns an integer.
|
|  __rpow__(self, value, mod=None, /)
|      Return pow(value, self, mod).
|
|  __rrshift__(self, value, /)
|      Return value>>self.
|
|  __rshift__(self, value, /)
|      Return self>>value.
|
|  __rsub__(self, value, /)
|      Return value-self.
|
|  __rtruediv__(self, value, /)
|      Return value/self.
|
|  __rxor__(self, value, /)
|      Return value^self.
|
|  __sizeof__(self, /)
|      Returns size in memory, in bytes.
|
|  __sub__(self, value, /)
|      Return self-value.
```

(suite sur la page suivante)

(suite de la page précédente)

```
|
|  __truediv__(self, value, /)
|      Return self/value.
|
|  __trunc__(…)
|      Truncating an Integral returns itself.
|
|  __xor__(self, value, /)
|      Return self^value.
|
|  as_integer_ratio(self, /)
|      Return integer ratio.
|
|      Return a pair of integers, whose ratio is exactly equal to the original int
|      and with a positive denominator.
|
|      >>> (10).as_integer_ratio()
|      (10, 1)
|      >>> (-10).as_integer_ratio()
|      (-10, 1)
|      >>> (0).as_integer_ratio()
|      (0, 1)
|
|  bit_count(self, /)
|      Number of ones in the binary representation of the absolute value of self.
|
|      Also known as the population count.
|
|      >>> bin(13)
|      '0b1101'
|      >>> (13).bit_count()
|      3
|
|  bit_length(self, /)
|      Number of bits necessary to represent self in binary.
|
|      >>> bin(37)
|      '0b100101'
|      >>> (37).bit_length()
|      6
|
|  conjugate(…)
|      Returns self, the complex conjugate of any int.
|
|  to_bytes(self, /, length, byteorder, *, signed=False)
|      Return an array of bytes representing an integer.
|
|      length
|        Length of bytes object to use.  An OverflowError is raised if the
|        integer is not representable with the given number of bytes.
|      byteorder
|        The byte order used to represent the integer.  If byteorder is 'big',
```

(suite sur la page suivante)

(suite de la page précédente)

```
|        the most significant byte is at the beginning of the byte array.  If
|        byteorder is 'little', the most significant byte is at the end of the
|        byte array.  To request the native byte order of the host system, use
|        `sys.byteorder' as the byte order value.
|      signed
|        Determines whether two's complement is used to represent the integer.
|        If signed is False and a negative integer is given, an OverflowError
|        is raised.
|
|  ----------------------------------------------------------------------
|  Class methods defined here:
|
|  from_bytes(bytes, byteorder, *, signed=False) from builtins.type
|      Return the integer represented by the given array of bytes.
|
|      bytes
|        Holds the array of bytes to convert.  The argument must either
|        support the buffer protocol or be an iterable object producing bytes.
|        Bytes and bytearray are examples of built-in objects that support the
|        buffer protocol.
|      byteorder
|        The byte order used to represent the integer.  If byteorder is 'big',
|        the most significant byte is at the beginning of the byte array.  If
|        byteorder is 'little', the most significant byte is at the end of the
|        byte array.  To request the native byte order of the host system, use
|        `sys.byteorder' as the byte order value.
|      signed
|        Indicates whether two's complement is used to represent the integer.
|
|  ----------------------------------------------------------------------
|  Static methods defined here:
|
|  __new__(*args, **kwargs) from builtins.type
|      Create and return a new object.  See help(type) for accurate signature.
|
|  ----------------------------------------------------------------------
|  Data descriptors defined here:
|
|  denominator
|      the denominator of a rational number in lowest terms
|
|  imag
|      the imaginary part of a complex number
|
|  numerator
|      the numerator of a rational number in lowest terms
|
|  real
|      the real part of a complex number
```

De même pour les fonctions.

[6]:
```
def add(x, y):
    "additionne x et y"
    result = x + y
    return result
```

[7]:
```
print(type(add))
```

```
<class 'function'>
```

[8]:
```
help(add)
```

```
Help on function add in module __main__:

add(x, y)
    additionne x et y
```

[9]:
```
# max() est une fonction 'builtin'
help(max)
```

```
Help on built-in function max in module builtins:

max(…)
    max(iterable, *[, default=obj, key=func]) -> value
    max(arg1, arg2, *args, *[, key=func]) -> value

    With a single iterable argument, return its biggest item. The
    default keyword-only argument specifies an object to return if
    the provided iterable is empty.
    With two or more arguments, return the largest argument.
```

2.6.3 Qui est là ?

Pour une information plus "brute" sur l'objet : `dir()` et `vars()`

[10]:
```
# une list de string regroupant tous les noms (attribut/méthode) accessibles dans la␣
↪classe int
dir(int)
```

[10]:
```
['__abs__',
 '__add__',
 '__and__',
 '__bool__',
 '__ceil__',
 '__class__',
 '__delattr__',
 '__dir__',
 '__divmod__',
 '__doc__',
 '__eq__',
 '__float__',
 '__floor__',
 '__floordiv__',
```

(suite sur la page suivante)

(suite de la page précédente)

```
'__format__',
'__ge__',
'__getattribute__',
'__getnewargs__',
'__gt__',
'__hash__',
'__index__',
'__init__',
'__init_subclass__',
'__int__',
'__invert__',
'__le__',
'__lshift__',
'__lt__',
'__mod__',
'__mul__',
'__ne__',
'__neg__',
'__new__',
'__or__',
'__pos__',
'__pow__',
'__radd__',
'__rand__',
'__rdivmod__',
'__reduce__',
'__reduce_ex__',
'__repr__',
'__rfloordiv__',
'__rlshift__',
'__rmod__',
'__rmul__',
'__ror__',
'__round__',
'__rpow__',
'__rrshift__',
'__rshift__',
'__rsub__',
'__rtruediv__',
'__rxor__',
'__setattr__',
'__sizeof__',
'__str__',
'__sub__',
'__subclasshook__',
'__truediv__',
'__trunc__',
'__xor__',
'as_integer_ratio',
'bit_count',
'bit_length',
'conjugate',
```

(suite sur la page suivante)

(suite de la page précédente)

```
 'denominator',
 'from_bytes',
 'imag',
 'numerator',
 'real',
 'to_bytes']
```

[11]:
```
# un dict de string:valeur regroupant tous les noms (attribut/méthode) accessibles␣
↪dans la classe int et leurs valeurs
vars(int)
```

[11]:
```
mappingproxy({'__new__': <function int.__new__(*args, **kwargs)>,
              '__repr__': <slot wrapper '__repr__' of 'int' objects>,
              '__hash__': <slot wrapper '__hash__' of 'int' objects>,
              '__getattribute__': <slot wrapper '__getattribute__' of 'int' objects>,
              '__lt__': <slot wrapper '__lt__' of 'int' objects>,
              '__le__': <slot wrapper '__le__' of 'int' objects>,
              '__eq__': <slot wrapper '__eq__' of 'int' objects>,
              '__ne__': <slot wrapper '__ne__' of 'int' objects>,
              '__gt__': <slot wrapper '__gt__' of 'int' objects>,
              '__ge__': <slot wrapper '__ge__' of 'int' objects>,
              '__add__': <slot wrapper '__add__' of 'int' objects>,
              '__radd__': <slot wrapper '__radd__' of 'int' objects>,
              '__sub__': <slot wrapper '__sub__' of 'int' objects>,
              '__rsub__': <slot wrapper '__rsub__' of 'int' objects>,
              '__mul__': <slot wrapper '__mul__' of 'int' objects>,
              '__rmul__': <slot wrapper '__rmul__' of 'int' objects>,
              '__mod__': <slot wrapper '__mod__' of 'int' objects>,
              '__rmod__': <slot wrapper '__rmod__' of 'int' objects>,
              '__divmod__': <slot wrapper '__divmod__' of 'int' objects>,
              '__rdivmod__': <slot wrapper '__rdivmod__' of 'int' objects>,
              '__pow__': <slot wrapper '__pow__' of 'int' objects>,
              '__rpow__': <slot wrapper '__rpow__' of 'int' objects>,
              '__neg__': <slot wrapper '__neg__' of 'int' objects>,
              '__pos__': <slot wrapper '__pos__' of 'int' objects>,
              '__abs__': <slot wrapper '__abs__' of 'int' objects>,
              '__bool__': <slot wrapper '__bool__' of 'int' objects>,
              '__invert__': <slot wrapper '__invert__' of 'int' objects>,
              '__lshift__': <slot wrapper '__lshift__' of 'int' objects>,
              '__rlshift__': <slot wrapper '__rlshift__' of 'int' objects>,
              '__rshift__': <slot wrapper '__rshift__' of 'int' objects>,
              '__rrshift__': <slot wrapper '__rrshift__' of 'int' objects>,
              '__and__': <slot wrapper '__and__' of 'int' objects>,
              '__rand__': <slot wrapper '__rand__' of 'int' objects>,
              '__xor__': <slot wrapper '__xor__' of 'int' objects>,
              '__rxor__': <slot wrapper '__rxor__' of 'int' objects>,
              '__or__': <slot wrapper '__or__' of 'int' objects>,
              '__ror__': <slot wrapper '__ror__' of 'int' objects>,
              '__int__': <slot wrapper '__int__' of 'int' objects>,
              '__float__': <slot wrapper '__float__' of 'int' objects>,
              '__floordiv__': <slot wrapper '__floordiv__' of 'int' objects>,
              '__rfloordiv__': <slot wrapper '__rfloordiv__' of 'int' objects>,
```

(suite sur la page suivante)

(suite de la page précédente)

```
             '__truediv__': <slot wrapper '__truediv__' of 'int' objects>,
             '__rtruediv__': <slot wrapper '__rtruediv__' of 'int' objects>,
             '__index__': <slot wrapper '__index__' of 'int' objects>,
             'conjugate': <method 'conjugate' of 'int' objects>,
             'bit_length': <method 'bit_length' of 'int' objects>,
             'bit_count': <method 'bit_count' of 'int' objects>,
             'to_bytes': <method 'to_bytes' of 'int' objects>,
             'from_bytes': <method 'from_bytes' of 'int' objects>,
             'as_integer_ratio': <method 'as_integer_ratio' of 'int' objects>,
             '__trunc__': <method '__trunc__' of 'int' objects>,
             '__floor__': <method '__floor__' of 'int' objects>,
             '__ceil__': <method '__ceil__' of 'int' objects>,
             '__round__': <method '__round__' of 'int' objects>,
             '__getnewargs__': <method '__getnewargs__' of 'int' objects>,
             '__format__': <method '__format__' of 'int' objects>,
             '__sizeof__': <method '__sizeof__' of 'int' objects>,
             'real': <attribute 'real' of 'int' objects>,
             'imag': <attribute 'imag' of 'int' objects>,
             'numerator': <attribute 'numerator' of 'int' objects>,
             'denominator': <attribute 'denominator' of 'int' objects>,
             '__doc__': "int([x]) -> integer\nint(x, base=10) -> integer\n\nConvert␣
↪a number or string to an integer, or return 0 if no arguments\nare given.  If x is␣
↪a number, return x.__int__().  For floating point\nnumbers, this truncates towards␣
↪zero.\n\nIf x is not a number or if base is given, then x must be a string,\nbytes,␣
↪or bytearray instance representing an integer literal in the\ngiven base.  The␣
↪literal can be preceded by '+' or '-' and be surrounded\nby whitespace.  The base␣
↪defaults to 10.  Valid bases are 0 and 2-36.\nBase 0 means to interpret the base␣
↪from the string as an integer literal.\n>>> int('0b100', base=0)\n4"})
```

[12]:
```
# dir() et vars() sans argument sont aussi très utiles
help(dir)
```

```
Help on built-in function dir in module builtins:

dir(…)
    dir([object]) -> list of strings

    If called without an argument, return the names in the current scope.
    Else, return an alphabetized list of names comprising (some of) the attributes
    of the given object, and of attributes reachable from it.
    If the object supplies a method named __dir__, it will be used; otherwise
    the default dir() logic is used and returns:
      for a module object: the module's attributes.
      for a class object:  its attributes, and recursively the attributes
        of its bases.
      for any other object: its attributes, its class's attributes, and
        recursively the attributes of its class's base classes.
```

2.6.4 Exercice

``max()`` est fournie par défaut (builtin), il y en a d'autres.

A quoi sert la seule fonction "builtin" qui commence "b" et finie par "n" ?

Indices :

1. `dir()` nous donne les noms utilisé actuellement
2. `help(max)` nous dit d'où vient cette fonction
3. `dir([truc])` nous donne le *contenu* de `truc`

2.7 Les conteneurs de base

Il existe 3 types de conteneurs de base très utilisés en Python
- `list` : tableau dynamique
- `tuple` : tableau statique
- `dict` : tableau associatif

Ils partagent certaines opérations communes
- connaître la taille du conteneur

```
taille = len(conteneur)  # 'taille' est un int
```

- tester la présence d'un objet dans le conteneur

```
is_here = obj in conteneur  # 'is_here' est un bool
```

- accéder à un élément du conteneur

```
elem_at_index = conteneur[index]  # 'elem_at_index' peut avoir n'importe quel type
```

2.7.1 Tableau dynamique : `list`

[1]:
```
# '[]' pour créer littéralement une liste
ma_liste = [42, 'bonjour', 3.14]
```

[2]:
```
type(ma_liste)
```

[2]:
```
list
```

[3]:
```
ma_liste
```

[3]:
```
[42, 'bonjour', 3.14]
```

[4]:
```
# la liste vide
vide = []
```

[5]:
```
type(vide)
```

[5]:
```
list
```

[6]:
```
vide
```

[6]:
```
[]
```

[7]:
```
# taille d'une liste
len(ma_liste)
```

[7]:
```
3
```

[8]:
```
# tester la présence d'un objet
42 in ma_liste
```

[8]:
```
True
```

[9]:
```
# tester la présence d'un objet
'au revoir' in ma_liste
```

[9]:
```
False
```

[10]:
```
# accès aux éléments -> indexation par des entiers

# Python compte à partir de 0 !

print(ma_liste[0])
print(ma_liste[1])
print(ma_liste[2])
```

```
42
bonjour
3.14
```

[11]:
```
# attention à ne pas dépasser

ma_liste[12]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[11], line 3
      1 # attention à ne pas dépasser
----> 3 ma_liste[12]

IndexError: list index out of range
```

[12]:
```
# on peut modifier le contenu d'un tableau dynamique

print(ma_liste)

ma_liste[1] = 'au revoir'

ma_liste
```

```
[42, 'bonjour', 3.14]
```

[12]:
```
[42, 'au revoir', 3.14]
```

[13]:
```
# concatenation de listes
```

(suite sur la page suivante)

(suite de la page précédente)

```
# '+' et '*' marchent comme pour str

a = [1, 2, 3]
b = ['un', 'deux']

print(a + b)
print(a * 2)
print(3 * b)
```

```
[1, 2, 3, 'un', 'deux']
[1, 2, 3, 1, 2, 3]
['un', 'deux', 'un', 'deux', 'un', 'deux']
```

2.7.2 Tableau statique : `tuple`

[14]:
```
# '()' pour créer littéralement un tuple
mon_tuple = (42, 'bonjour', 3.14)
```

[15]:
```
type(mon_tuple)
```

[15]:
```
tuple
```

[16]:
```
mon_tuple
```

[16]:
```
(42, 'bonjour', 3.14)
```

[17]:
```
# le tuple vide
vide = ()
```

[18]:
```
type(vide)
```

[18]:
```
tuple
```

[19]:
```
vide
```

[19]:
```
()
```

[20]:
```
# taille d'un tuple
len(mon_tuple)
```

[20]:
```
3
```

[21]:
```
# tester la présence d'un objet
42 in mon_tuple
```

[21]:
```
True
```

[22]:
```
# tester la présence d'un objet
'au revoir' in mon_tuple
```

[22]:
```
False
```

[23]:
```
# accès aux éléments -> indexation par des entiers

# Python compte à partir de 0 !

print(mon_tuple[0])
print(mon_tuple[1])
print(mon_tuple[2])
```

```
42
bonjour
3.14
```

[24]:
```
# attention à ne pas dépasser
mon_tuple[12]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[24], line 2
      1 # attention à ne pas dépasser
----> 2 mon_tuple[12]

IndexError: tuple index out of range
```

[25]:
```
# on NE PEUT PAS modifier le contenu d'un tableau statique
mon_tuple[1] = 'au revoir'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[25], line 2
      1 # on NE PEUT PAS modifier le contenu d'un tableau statique
----> 2 mon_tuple[1] = 'au revoir'

TypeError: 'tuple' object does not support item assignment
```

[26]:
```
# concatenation de tuples avec '+' et '*'

a = (1, 2, 3)
b = ('un', 'deux')

print(a + b)
print(a * 2)
print(3 * b)
```

```
(1, 2, 3, 'un', 'deux')
(1, 2, 3, 1, 2, 3)
('un', 'deux', 'un', 'deux', 'un', 'deux')
```

Quelques subtilités

[27]:
```
# '()' pour créer littéralement un tuple ...
# ... enfin, pas tout à fait
```

(suite sur la page suivante)

(suite de la page précédente)

```
a = (12)

print("qu'est-ce que 'a' ?")
print(type(a))
print(a)
print()

b = 42, 'bonjour', 3.14

print("qu'est-ce que 'b' ?")
print(type(b))
print(b)
```

```
qu'est-ce que 'a' ?
<class 'int'>
12

qu'est-ce que 'b' ?
<class 'tuple'>
(42, 'bonjour', 3.14)
```

[28]:
```
# le 1-tuple

tuple_de_taille_1 = (1,)  # notez la virgule
print(type(tuple_de_taille_1))
print(tuple_de_taille_1)
```

```
<class 'tuple'>
(1,)
```

2.7.3 Tableau associatif : `dict`

[29]:
```
# '{clef: valeur}' pour créer littéralement un dict ...
mon_dict = {'un': 42, 2: 'bonjour', 'trois': 3.14}
```

[30]:
```
type(mon_dict)
```

[30]:
```
dict
```

[31]:
```
mon_dict
```

[31]:
```
{'un': 42, 2: 'bonjour', 'trois': 3.14}
```

[32]:
```
# le dict vide
vide = {}
```

[33]:
```
type(vide)
```

[33]:
```
dict
```

[34]:
```
vide
```

[34]:
```
{}
```

[35]:
```
# taille d'un dict
len(mon_dict)
```

[35]:
```
3
```

[36]:
```
# tester la présence d'une clef
'un' in mon_dict
```

[36]:
```
True
```

[37]:
```
# tester la présence d'une clef
2 in mon_dict
```

[37]:
```
True
```

[38]:
```
# tester la présence d'une clef
42 in mon_dict
```

[38]:
```
False
```

[39]:
```
# tester la présence d'une clef
'bonjour' in mon_dict
```

[39]:
```
False
```

[40]:
```
# si on tient vraiment à chercher dans les valeurs (et non les clefs)
'bonjour' in mon_dict.values()
```

[40]:
```
True
```

[41]:
```
# accès aux éléments -> indexation par des clefs
print(mon_dict['un'])
print(mon_dict[2])
print(mon_dict['trois'])
```

```
42
bonjour
3.14
```

[42]:
```
# attention à l'absence de clef
mon_dict['toto']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[42], line 2
      1 # attention à l'absence de clef
----> 2 mon_dict['toto']

KeyError: 'toto'
```

[43]:
```
# on peut modifier le contenu d'un tableau dynamique
# quand une clef existe déjà :
```

(suite sur la page suivante)

(suite de la page précédente)

```
print(mon_dict)
mon_dict['trois'] = 2.71828
mon_dict
```

```
{'un': 42, 2: 'bonjour', 'trois': 3.14}
```

[43]:
```
{'un': 42, 2: 'bonjour', 'trois': 2.71828}
```

[44]:
```
# on peut modifier le contenu d'un tableau dynamique
# avec une nouvelle clef :
print(mon_dict)
mon_dict['quatre'] = "nouvelle case"
mon_dict
```

```
{'un': 42, 2: 'bonjour', 'trois': 2.71828}
```

[44]:
```
{'un': 42, 2: 'bonjour', 'trois': 2.71828, 'quatre': 'nouvelle case'}
```

/ !\ l'instruction `mon_dict[clef] = valeur` marchera toujours*, la taille du dictionnaire *peut* augmenter cela la situation.
(*) *presque* toujours, cf exercice

[45]:
```
# concatenation de dict

a = {'un': 1, 'deux': 2}
b = {'trois': 3}

a | b
```

[45]:
```
{'un': 1, 'deux': 2, 'trois': 3}
```

2.7.4 Une chaîne de caractères est un tableau statique

On retrouve `len()`, `in`, et l'indexation

[46]:
```
mot = 'python'
```

[47]:
```
len(mot)
```

[47]:
```
6
```

[48]:
```
'y' in mot
```

[48]:
```
True
```

[49]:
```
print(mot[0], mot[1], mot[2])
print(mot[3], mot[4], mot[5])
```

```
p y t
h o n
```

[50]:
```
# comme tuple, c'est un tableau statique
mot[0] = 'P'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[50], line 2
      1 # comme tuple, c'est un tableau statique
----> 2 mot[0] = 'P'

TypeError: 'str' object does not support item assignment
```

2.7.5 Conversion en booléen

Les objets de type `list`, `tuple`, `dict`, ainsi que tous les autres types de conteneurs, se convertissent en booléen :

```
Faux si conteneur vide, Vrai sinon
```

[51]:
```
# list
vide = []
pas_vide = [1, 2, 3]

print(vide, bool(vide))
print(pas_vide, bool(pas_vide))
```

```
[] False
[1, 2, 3] True
```

[52]:
```
# tuple
vide = ()
pas_vide = (1, 2, 3)

print(vide, bool(vide))
print(pas_vide, bool(pas_vide))
```

```
() False
(1, 2, 3) True
```

[53]:
```
# dict
vide = {}
pas_vide = {1: 'un', 2: 'deux', 3: 'trois'}

print(vide, bool(vide))
print(pas_vide, bool(pas_vide))
```

```
{} False
{1: 'un', 2: 'deux', 3: 'trois'} True
```

2.7.6 Conversion entre conteneurs

[54]:
```
ma_liste = [1, 2, 3]
mon_tuple = (10, 20, 30)
mon_dict = {'a': 100, 'b': 200, 'c': 300}
```

[55]:
```
# list -> tuple
tuple(ma_liste)
```

[55]:
```
(1, 2, 3)
```

[56]:
```
# tuple -> list
list(mon_tuple)
```

[56]:
```
[10, 20, 30]
```

[57]:
```
# dict -> list
list(mon_dict)
```

[57]:
```
['a', 'b', 'c']
```

[58]:
```
# list -> dict
clef_valeur = [['a', 100], ['b', 200], ['c', 300]]
dict(clef_valeur)
```

[58]:
```
{'a': 100, 'b': 200, 'c': 300}
```

2.7.7 Exercice

Construire des dictionnaires utilisant des clefs de différents types.

Quels types ne sont pas autorisés ?

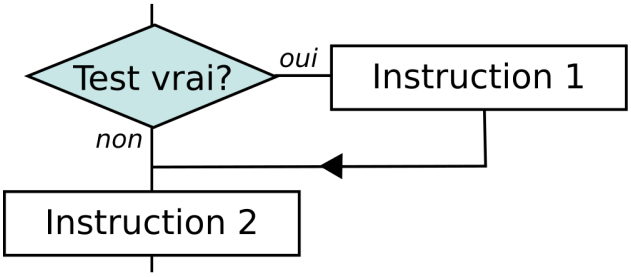
— *Vecteur de 0*

2.8 Structure de contrôle

— if (test) elif else
  — conversion implicite en booléen
— while (test) [break/continue] else
— for (iter) [break/continue] else

susucre : `a = x if cond else y`

2.8.1 IF - ELIF - ELSE



Structure de `if - elif -else`

```
# instruction avant le if

if test_1:
```

(suite sur la page suivante)

(suite de la page précédente)

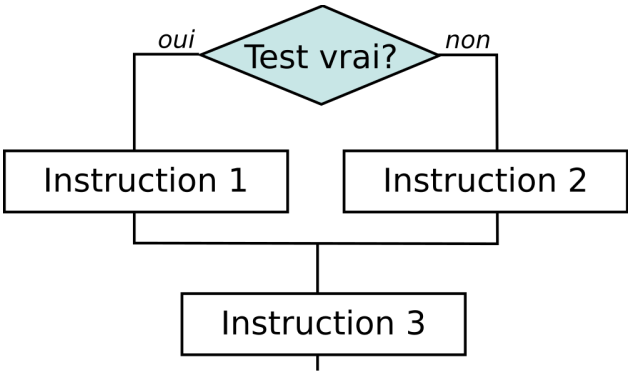
```
    # instruction A
elif test_2:
    # instruction B
elif test_3:
    # instruction C
else:
    # instruction D

# instruction après le if
```

[1]:
```
print('start')

a = 1

if a == 1:
    print('travail')

print('end')
```

```
start
travail
end
```

[2]:
```
print('start')

a = 2

if a == 1:
    print('travail')

print('end')
```

```
start
end
```



[3]:
```
print('start')

a = 1

if a == 1:
```

(suite sur la page suivante)

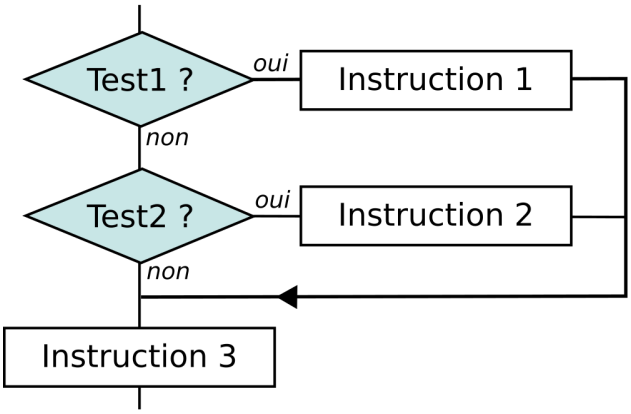
(suite de la page précédente)

```
    print('travail 1')
else:
    print('travail 2')

print('end')
```

```
start
travail 1
end
```

[4]:
```
print('start')

a = 1

if a == 2:
    print('travail 1')
else:
    print('travail 2')

print('end')
```

```
start
travail 2
end
```



[5]:
```
a = 1

if a == 1:
    print('travail 1')
elif a == 2:
    print('travail 2')
else:
    print('dodo')
```

```
travail 1
```

[6]:
```
a = 2

if a == 1:
```

(suite sur la page suivante)

(suite de la page précédente)

```
    print('travail 1')
elif a == 2:
    print('travail 2')
else:
    print('dodo')
```

```
travail 2
```

[7]:
```
a = 3

if a == 1:
    print('travail 1')
elif a == 2:
    print('travail 2')
else:
    print('dodo')
```

```
dodo
```

2.8.2 FOR

En python, `for` permet d'itérer sur les éléments d'une séquence (ex : `list`, `tuple`, `dict`).

[8]:
```
ma_liste = ['toto', 42, 3.14]

for elem in ma_liste:
    print(elem)
```

```
toto
42
3.14
```

[9]:
```
mon_dict = {'un': 42, 2: 'bonjour', 'trois': 3.14}

for elem in mon_dict:
    print(elem)
```

```
un
2
trois
```

[10]:
```
for elem in 'python':
    print(elem)
```

```
p
y
t
h
o
n
```

Structure de `for - continue / break - else`

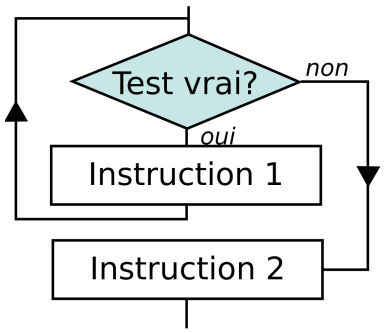
```
# instruction avant le for

for elem in sequence:
    # debut des instructions
    if test_break:
        # on arrete tout
        break
    if test_continue:
        # on passe à l'itération suivante
        continue
    # suite des instructions
    # pas vue si continue
else:
    # executer si on est sorti de la boucle sans break

# instruction après le for
```

[11]:
```
#liste = range(15)
liste = range(10)
result = []

for elem in liste:
    if elem > 10:
        result = 'interdit de dépasser 10'
        break
    if elem % 2:
        # ignorer les impairs
        continue
    result.append(elem)  # un peu comme 'result += [elem]' mais bien mieux
else:
    print("tout s'est bien passé")

print(result)
```

```
tout s'est bien passé
[0, 2, 4, 6, 8]
```

Remarque Ce dernier exemple présente de gros défauts :
- changer la nature de `result` selon le contexte
- `else: print(...)` est un mode de communication très faible

2.8.3 WHILE



Structure de `while - continue / break - else`

```
# instruction avant le while

while test_boucle:
    # debut des instructions
    if test_break:
        # on arrete tout
        break
    if test_continue:
        # on passe à l'itération suivante
        continue
    # suite des instructions
    # pas vue si continue
else:
    # executer si on est sorti de la boucle sans break

# instruction après le while
```

[12]:
```
# compteur

a = 0

while a < 10:
    a += 1
    print(a, end=' ')
```

```
1 2 3 4 5 6 7 8 9 10
```

[13]:
```
# compteur pair

a = 0

while a < 10:
    a += 1
    if a % 2:
        continue
    print(a, end=' ')
```

```
2 4 6 8 10
```

[14]:
```
def indice(liste, valeur):
    "renvoie le premier indice tq liste[indice] == valeur"
    i = 0

    while i < len(liste):
        if liste[i] == valeur:
            result = i
            break
        i += 1
    else:
        result = None

    return result
```

(suite sur la page suivante)

(suite de la page précédente)

```
ma_liste = ['zero', 'un', 'deux', 'trois']
```

[15]:
```
indice(ma_liste, 'deux')
```

[15]:
```
2
```

[16]:
```
indice(ma_liste, 'quatre')
```

En pratique, on préférera l'utilisation du `for` (éventuellement couplé à un `if`) qui ne risque pas de partir dans des boucles infinies.

Par exemple, le code suivant tournera à l'infini (et on n'aime pas ça) :

```
a = 0

while a < 10:
    if a % 2:
        continue
    print(a, end=' ')
    a += 1
```

[17]:
```
# compteur pair

for a in range(25):
    if a > 10:
        break
    if a % 2:
        continue
    print(a, end=' ')
```

```
0 2 4 6 8 10
```

2.8.4 Exercice

- *Conversion de secondes (étape 2)*
- *Compte occurences (étape 1)*
- *Suite de Collatz*

2.9 Structuration du code

2.9.1 Bloc et indentation

Les fonctions et les structures de contrôle introduisent l'idée de « bloc d'instructions ».

Elles prennent toutes la même forme :

```
MotClef ... :
    code
    code
    code
```

Le “:” est à chaque fois suivi d’une **indentation**.

C’est l’indentation qui signifie à l’interpréteur où commence et finit le bloc.

En Python, l’indentation est signifiante :
— chaque ligne d’un bloc doit avoir la même indentation
— on ne doit pas mélanger *espaces* et *tabulations* !

2.9.2 Saut de ligne

Une ligne correspond à une instruction. Il n’y a pas de symbole signifiant la fin d’une « ligne d’instruction ». (eg “;” en C)

Mais parfois une ligne est trop courte pour une instruction. La solution est alors d’utiliser “`()`”, “`[]`” ou “`{}`” sur plusieurs lignes.

[1]:
```
# une liste sur plusieurs lignes
l = [
    1,
    2,
    3,
    4,  # la dernière virgule est autorisée et ne change rien
]
l
```

[1]:
```
[1, 2, 3, 4]
```

[2]:
```
# un dictionnaire sur plusieurs lignes
d = {
    'un': 1,
    'deux': 2,
    'trois': 3,
    'quatre': 4
}
d
```

[2]:
```
{'un': 1, 'deux': 2, 'trois': 3, 'quatre': 4}
```

[3]:
```
# fermer une parenthèse de fonction sur une autre ligne
nom_de_fonction_vraiment_trop_long_pour_tout_ecrire_sur_une_ligne = max

result = nom_de_fonction_vraiment_trop_long_pour_tout_ecrire_sur_une_ligne(
    1, 2, 3, 4
)

result
```

[3]:
```
4
```

[4]:
```
# un calcul trop long
a = 1.
b = 2.
c = 3.
d = 4.

result = (a + (b ** 3) * 2) / (((c + d) * (a + b)) + ((a + b + c + d) * (a + b + c) *␣
```

(suite sur la page suivante)

(suite de la page précédente)

```
↪(a + b) * a))

result
```

[4]:
```
0.0845771144278607
```

[5]:
```
# c'est l'occasion de structurer un peu tout ça (si possible...)
a = 1.
b = 2.
c = 3.
d = 4.

result = (
    (a + (b ** 3) * 2)
    /
    (
        (c + d) * (a + b)
        +
        (a + b + c + d) * (a + b + c) * (a + b) * a
    )
)

result
```

[5]:
```
0.0845771144278607
```

PS : il existe un jeu de caractères pour indiquer la continuation de ligne, mais je n'en parlerai pas.

2.9.3 Exercices

2.10 Lire les erreurs

2.10.1 Qu'est-ce c'est ?

— L'erreur apparait au moment où l'instruction erronée est exécutée dans le cours du programme.
— A priori, l'apparition d'une erreur interrompt le programme immédiatement -> plantage.
— (il est quand même possible de *gérer les exceptions*)

2.10.2 Les informations fournies par une erreur

— le type d'erreur
— le message d'erreur
— la ligne de code en jeu
— la pile d'appel -> *traceback*

Coder == Débugger, autant utiliser l'information qu'on nous donne.

2.10.3 Exemples classiques d'erreurs

NameError

[1]:
```
nexistepas
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[1], line 1
----> 1 nexistepas

NameError: name 'nexistepas' is not defined
```

ZeroDivisionError

```
[2]: 2 / 0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Cell In[2], line 1
----> 1 2 / 0

ZeroDivisionError: division by zero
```

TypeError

```
[3]: 'un' + 2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[3], line 1
----> 1 'un' + 2

TypeError: can only concatenate str (not "int") to str
```

IndexError

```
[4]: hey = 'bonjour'
     hey[9]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[4], line 2
      1 hey = 'bonjour'
----> 2 hey[9]

IndexError: string index out of range
```

KeyError

```
[5]: foobar={'un': 1}
     foobar['deux']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[5], line 2
      1 foobar={'un': 1}
----> 2 foobar['deux']
```

(suite sur la page suivante)

(suite de la page précédente)

```
KeyError: 'deux'
```

2.10.4 *Traceback* et imbrication d'appels

La *Traceback* permet de suivre l'imbrication des appels au moment de l'erreur.

[6]:
```
def f0(x):
    return x / 0

def f1(x):
    return f0(x) / 1

def f2(x):
    return f1(x) / 2

def f3(x):
    return f2(x) / 3

f3(1)
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Cell In[6], line 13
     10 def f3(x):
     11     return f2(x) / 3
---> 13 f3(1)

Cell In[6], line 11, in f3(x)
     10 def f3(x):
---> 11     return f2(x) / 3

Cell In[6], line 8, in f2(x)
      7 def f2(x):
----> 8     return f1(x) / 2

Cell In[6], line 5, in f1(x)
      4 def f1(x):
----> 5     return f0(x) / 1

Cell In[6], line 2, in f0(x)
      1 def f0(x):
----> 2     return x / 0

ZeroDivisionError: division by zero
```

2.10.5 Exercices

3 Aller plus loin avec Python

3.1 Utilisation avancée des chaines de caractères

3.1.1 Formatted string literals

Plus souvent appelés f-string (https://docs.python.org/3/tutorial/inputoutput.html#formatted-string-literals), ils sont un moyen simple et efficace de produire des chaines formatées.

[1]:
```
a = 2
f"*~ {a} ~*"  # f-string !
```

[1]:
```
'*~ 2 ~*'
```

[2]:
```
a = 'toto'
f"*~ {a} ~*"  # le même f-string mais la variable a une autre valeur
```

[2]:
```
'*~ toto ~*'
```

Champs multiples

[3]:
```
a = 1
b = 2
f"~{a}~  -{b}-"
```

[3]:
```
'~1~  -2-'
```

[4]:
```
a = 'toto'
b = 'tata'
f"~{a}~  -{b}-"
```

[4]:
```
'~toto~  -tata-'
```

[5]:
```
a = 42
b = 'tata'
f"~{a}~  -{b}-"
```

[5]:
```
'~42~  -tata-'
```

Mise en forme

Des instructions de formatage peuvent être ajoutées `{variable:formatage}`

[6]:
```
# On peut justifier et centrer un champ
a = 'toto'
b = 'tata'
c = 'titi'
f"| {a:>10} | {b:^10} | {c:<10} |"
```

[6]:
```
'|       toto |    tata    | titi       |'
```

[7]:
```
# On peut controler l'affichage des nombres
a = 5
print(f"{a}")
print(f"{a:3}")
print(f"{a:03}")
```

```
5
  5
005
```

[8]:
```
# et des nombres flottants
a = 5
pi = 3.14159
print(f"{pi}")       # sans formatage, affichage au plus près de la donnée
print(f"{a}")        # sans formatage, affichage au plus près de la donnée
print(f"{a:f}")      # affichons un int comme un float
print(f"{a:.3f}")    # controlons le nombre de décimales
print(f"{pi:.2f}")   # controlons le nombre de décimales
print(f"{a:e}")      # notation scientifique
print(f"{a:.2e}")    # notation scientifique avec 2 décimales
```

```
3.14159
5
5.000000
5.000
3.14
5.000000e+00
5.00e+00
```

3.1.2 Méthode des `str`

[9]:
```
# en python, les chaines de caractères disposent de nombreuses méthodes

str_methods = [name for name in dir(str) if name[0] != '_']
for name in str_methods:
    print(f'str.{name}(...)')
```

```
str.capitalize(…)
str.casefold(…)
str.center(…)
str.count(…)
str.encode(…)
str.endswith(…)
```

(suite sur la page suivante)

(suite de la page précédente)

```
str.expandtabs(…)
str.find(…)
str.format(…)
str.format_map(…)
str.index(…)
str.isalnum(…)
str.isalpha(…)
str.isascii(…)
str.isdecimal(…)
str.isdigit(…)
str.isidentifier(…)
str.islower(…)
str.isnumeric(…)
str.isprintable(…)
str.isspace(…)
str.istitle(…)
str.isupper(…)
str.join(…)
str.ljust(…)
str.lower(…)
str.lstrip(…)
str.maketrans(…)
str.partition(…)
str.removeprefix(…)
str.removesuffix(…)
str.replace(…)
str.rfind(…)
str.rindex(…)
str.rjust(…)
str.rpartition(…)
str.rsplit(…)
str.rstrip(…)
str.split(…)
str.splitlines(…)
str.startswith(…)
str.strip(…)
str.swapcase(…)
str.title(…)
str.translate(…)
str.upper(…)
str.zfill(…)
```

[10]:
```
des_mots = ['Bonjour', 'tout', 'le', 'monde', '!']
mot = 'bidule'
phrase = "Vive les tartes a la myrtille."
paragraphe = '''
B: What is your name?
A: It is Arthur, King of the Britons.
B: What is your quest?
A: To seek the Holy Grail.
B: What is the air-speed velocity of a swallow?
A: What do you mean?
```

(suite sur la page suivante)

(suite de la page précédente)

```
B: Er ... I don't know that ... Aaaaarrrrrrggghhh!
'''
```

3.1.3 Tester le contenu

[11]:
```
# tester la présence d'une sous chaine
print('le' in des_mots)  # élement dans une liste
print('le' in mot)
print('le' in phrase)  # sous chaine
```

```
True
True
True
```

[12]:
```
# tester le début
print(mot.startswith('b'))
print(mot.startswith('bi'))
print(mot.startswith('ba'))
```

```
True
True
False
```

[13]:
```
# tester la fin
print(phrase.endswith('.'))
print(phrase.endswith('myrtille'))
```

```
True
False
```

[14]:
```
# tester la nature du contenu
[name for name in dir(str) if name.startswith('is')]
```

[14]:
```
['isalnum',
 'isalpha',
 'isascii',
 'isdecimal',
 'isdigit',
 'isidentifier',
 'islower',
 'isnumeric',
 'isprintable',
 'isspace',
 'istitle',
 'isupper']
```

3.1.4 Compter / localiser

[15]:
```
# str.count()

print(paragraphe.count('?'))
print(paragraphe.count('the'))
```

```
4
3
```

[16]:
```
# str.index()

print(phrase.index('a'))
print(phrase.index('la'))
print(phrase.rindex('a'))
```

```
10
18
19
```

3.1.5 Fusionner, séparer

[17]:
```
# str.join()

fatiguant = ''
for elem in des_mots:
    if fatiguant:
        fatiguant += ' ' + elem
    else:
        fatiguant += elem
repr(fatiguant)
```

[17]:
```
"'Bonjour tout le monde !'"
```

[18]:
```
facile = ' '.join(des_mots)
repr(facile)
```

[18]:
```
"'Bonjour tout le monde !'"
```

[19]:
```
# str.split()
phrase.split()
```

[19]:
```
['Vive', 'les', 'tartes', 'a', 'la', 'myrtille.']
```

[20]:
```
# str.split()
phrase.split('t')
```

[20]:
```
['Vive les ', 'ar', 'es a la myr', 'ille.']
```

[21]:
```
# str.splitlines()
paragraphe.splitlines()
```

[21]:
```
['',
 'B: What is your name?',
 'A: It is Arthur, King of the Britons.',
 'B: What is your quest?',
 'A: To seek the Holy Grail.',
 'B: What is the air-speed velocity of a swallow?',
 'A: What do you mean? ',
 "B: Er … I don't know that … Aaaaarrrrrrggghhh!"]
```

[22]:
```
# str.partition()
phrase.partition('a')
```

[22]:
```
('Vive les t', 'a', 'rtes a la myrtille.')
```

[23]:
```
# str.partition()
phrase.partition('tartes')
```

[23]:
```
('Vive les ', 'tartes', ' a la myrtille.')
```

[24]:
```
# str.rpartition() pour partir de la droite
phrase.rpartition('a')
```

[24]:
```
('Vive les tartes a l', 'a', ' myrtille.')
```

3.1.6 Mise en forme

[25]:
```
# Changer la casse

print(phrase.lower())
print(phrase.upper())
print(phrase.title())
print(phrase.capitalize())
print(mot, mot.capitalize())
```

```
vive les tartes a la myrtille.
VIVE LES TARTES A LA MYRTILLE.
Vive Les Tartes A La Myrtille.
Vive les tartes a la myrtille.
bidule Bidule
```

[26]:
```
# justifier, centrer

print(repr(mot.rjust(10)))
print(repr(mot.ljust(10)))
print(repr(mot.center(10)))
```

```
'    bidule'
'bidule    '
'  bidule  '
```

3.1.7 Exercices

— *Conversion de secondes (étape 3)*

```
exercice = '''
    Définir un texte comme celui-ci et le découper en une liste de mots.
    Utiliser la fonction `compte_occurrence()` pour compter le nombre
    d'occurrences de chaque mot présent dans le texte.
'''
```

3.2 Un point sur les méthodes

3.2.1 *Déjà vu*

```
ma_liste.append(42)
```

Cette expression a pour effet d'agrandir `ma_liste` en ajoutant à la fin une case contenant la valeur `42`.
On voit ici que

- `ma_liste` est une liste
- `ma_liste.append` est une fonction
- `ma_liste.append(42)` est l'appel de cette fonction.

La fonction s'exécute en connaissant `ma_liste` **et** `42`.

3.2.2 La syntaxe des méthodes

La syntaxe générale se présente ainsi :

```
mon_objet.nom_methode(argument)
```

`mon_objet` est un objet quelconque. On y accède le plus souvent par un nom de variable mais parfois aussi directement par une expression littérale.

```
# ma_liste est un nom de variable qui désigne un liste
ma_liste.extend(des_mots)
```

```
# ", " est une expression littérale de string
", ".join(des_mots)
```

`nom_methode` est un nom, comme un nom de variable. Les noms disponibles dépendent du type de `mon_objet`.

```
# python peut vous renseigner sur les méthodes disponibles pour un objet donné
dir(mon_objet)
help(mon_objet)
```

`argument` représente les arguments passés à la fonction. Ils suivent les mêmes règles que pour une fonction classique.

```
mon_objet.nom_methode(a, b)
mon_objet.nom_methode(a, b, option=c)
```

3.2.3 Les attributs

En vérité, un objet peut abriter derrière le `.` des fonctions (méthodes) mais aussi des valeurs (attributs).
Les attributs disponibles et leurs noms dépendent entièrement du type de l'objet.

```
help(mon_objet)
```

3.2.4 Conclusion

Selon leur type, les objets peuvent exposer derrière le `.` un ou plusieurs attributs et méthodes.
Les noms des méthodes/attributs est fixe pour un type donné.
Il faut se rapporter à la documentation du type de l'objet pour découvrir ce que contient qu'il propose.

Quand une méthode est appelée, elle connait à la fois :
- l'objet auquel elle se rattache (à gauche du `.`)
- les arguments de la fonction (dans les parenthèses)

3.2.5 Exercices

Découvrir quelques méthodes des types de bases :
- `list.append()`
- `dict.get()`
- `str.split()`

3.3 Utilisation avancée des indices

3.3.1 Indices négatifs

[1]:
```
x = 'indice'

print(x[-1])
print(x[-2])
print(x[-3])
print(x[-4])
print(x[-5])
print(x[-6])
```

```
e
c
i
d
n
i
```

3.3.2 A slice of ham

Syntaxe de l'indexation par tranche :

```
conteneur[start:stop:step]  # de start à stop par saut de step
conteneur[start:stop]       # step = 1
conteneur[start:]           # de start à la fin
conteneur[:stop]            # du début à stop
```

Attention :
- `start` est inclue
- `stop` est exclue

[2]:
```
x = list(range(10))
x
```

[2]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

[3]:
```
x[2:8:3]
```

[3]:
```
[2, 5]
```

[4]:
```
x[:4]
```

[4]:
```
[0, 1, 2, 3]
```

[5]:
```
x[4:]
```

[5]:
```
[4, 5, 6, 7, 8, 9]
```

[6]:
```
x[::-1]
```

[6]:
```
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

[7]:
```
x[:-2]
```

[7]:
```
[0, 1, 2, 3, 4, 5, 6, 7]
```

3.3.3 Affectation par slice

[8]:
```
x
```

[8]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

[9]:
```
x[::2] = ['X'] * 5
x
```

[9]:
```
['X', 1, 'X', 3, 'X', 5, 'X', 7, 'X', 9]
```

3.3.4 Exercices

Ecrire le corps de la fonction suivante :

```
def construire_matrice(valeurs, nb_lignes):
    ''' renvoie la matrice composée de `valeurs` à `nb_lignes` lignes.

    Une matrice est une liste de listes.
    `valeurs` est une liste contenant les valeurs de la matrice : d'abord la 1ère␣
↪ligne, puis la seconde, ...
    `nb_lignes` est un entier indiquant le nombre de lignes de la matrice retournée.
    '''
```

Enrichir cette fonction avec un nouvel argument : `construire_matrice(valeurs, nb_lignes, valeurs_rangees_par_lignes=True)`

3.4 Unpacking

[1]:
```
mes_donnees = ('Mickey', 'Mouse')
```

[2]:
```
# ok mais un fastidieux
prenom = mes_donnees[0]
nom = mes_donnees[1]

nom, prenom
```

[2]:
```
('Mouse', 'Mickey')
```

[3]:
```
# mieux : unpacking !
prenom, nom = mes_donnees

nom, prenom
```

[3]:
```
('Mouse', 'Mickey')
```

[4]:
```
a, b, c = 'un', 'deux', 'trois'

print(a)
print(b)
print(c)
```

```
un
deux
trois
```

[5]:
```
# échanger deux variables

# calcul de la suite de Fibonacci
a, b = 0, 1
while b < 100:
    a, b = b, a+b
    print(b, end=' ')
```

```
1 2 3 5 8 13 21 34 55 89 144
```

3.4.1 Exercices

3.5 Le retour des fonctions

3.5.1 Retourner plusieurs valeurs

[1]:
```
def compte_sur_une_main():
    return 1, 2, 3, 4, 5

result = compte_sur_une_main()

type(result), result
```

[1]:
```
(tuple, (1, 2, 3, 4, 5))
```

[2]:
```
# on peut utiliser l'unpacking pour nommer directement les résultats

def double_triple(x):
```

(suite sur la page suivante)

(suite de la page précédente)

```
    return 2 * x, 3 * x

a = 'lo'

double_a, triple_a = double_triple(a)

print(double_a)
print(triple_a)
```

```
lolo
lololo
```

3.5.2 Nommer les arguments

[3]:
```
# Les noms des arguments font partie de l'interface d'une fonction

def divise(numerateur, denominateur):
    "calcul la division de numerateur par denominateur"
    return numerateur / denominateur

help(divise)
```

```
Help on function divise in module __main__:

divise(numerateur, denominateur)
    calcul la division de numerateur par denominateur
```

[4]:
```
divise(10, 2)
```

[4]:
```
5.0
```

[5]:
```
divise(numerateur=10, denominateur=2)
```

[5]:
```
5.0
```

[6]:
```
divise(denominateur=2, numerateur=10)
```

[6]:
```
5.0
```

3.5.3 Arguments avec une valeur par défaut

[7]:
```
def indente(texte, indentation='    '):
    return indentation + texte

print('toto')
print(indente('toto'))
print(indente('toto', '- - '))
print(indente('toto', indentation='~~~~'))
```

```
toto
    toto
```

(suite sur la page suivante)

(suite de la page précédente)

```
- - toto
~~~~toto
```

3.5.4 Nombre d'arguments variables

[8]:
```
# En pratique

def somme(*args):
    "calcule la somme de tous les arguments"
    result = 0
    for valeur in args:
        result += valeur
    return result
```

[9]:
```
somme(2, 2)
```

[9]:
```
4
```

[10]:
```
somme(2, 5, 78)
```

[10]:
```
85
```

[11]:
```
# `args` est un tuple

def test(*args):
    print(type(args), args)

test(2, 2)
test(2, 5, 78)
```

```
<class 'tuple'> (2, 2)
<class 'tuple'> (2, 5, 78)
```

3.5.5 Utiliser un list/tuple pour passer les arguments d'une fonction

[12]:
```
# une fonction avec plusieurs arguments

def affiche_abc(a, b, c):
    print('voici a :', a)
    print('voici b :', b)
    print('voici c :', c)
```

[13]:
```
# 'arguments' contient mes arguments

arguments = ['^ ^', ' o ', '\_/']
```

[14]:
```
# si l'on sait comme ici que 'affiche_abc()' prend 3 arguments
# on peut faire ainsi

affiche_abc(arguments[0], arguments[1], arguments[2])
```

```
voici a : ^ ^
voici b :  o
voici c : \_/
```

[15]:
```
# voici une solution plus courte et robuste

affiche_abc(*arguments)
```

```
voici a : ^ ^
voici b :  o
voici c : \_/
```

3.5.6 Nombre d'arguments "mot-clef" variables

[16]:
```
# `kwargs` est un dict

def test(**kwargs):
    print(type(kwargs), kwargs)

test(a=2, b=2)
test(toto=2, tata=5, titi=78)
```

```
<class 'dict'> {'a': 2, 'b': 2}
<class 'dict'> {'toto': 2, 'tata': 5, 'titi': 78}
```

3.5.7 Utiliser un dict pour passer les arguments d'une fonction

[17]:
```
# gardons notre fonction avec plusieurs arguments

def affiche_abc(a, b, c):
    print('voici a :', a)
    print('voici b :', b)
    print('voici c :', c)
```

[18]:
```
# cette fois, nos arguments sont stockés dans un dict

keyword_arguments = {
'a' : "* *",
'b' : " ' ",
'c' : " ~ ",
}
```

[19]:
```
# si on connait bien les mots-clefs

affiche_abc(a=keyword_arguments['a'], b=keyword_arguments['b'], c=keyword_arguments['c
↪'])
```

```
voici a : * *
voici b :  '
voici c :  ~
```

[20]:
```
# si on connait la bonne formule

affiche_abc(**keyword_arguments)
```

```
voici a : * *
voici b :  '
voici c :  ~
```

3.5.8 fonction anonyme : `lambda`

— Pour définir rapidement une fonction simple.
— Le traitement des arguments est le même que pour les fonctions.
— Mais le corps est réduit à une instruction dont le résultat est la valeur retournée.

[21]:
```
double = lambda x: 2 * x

type(double)
```

[21]:
```
function
```

[22]:
```
double(3)
```

[22]:
```
6
```

[23]:
```
double('to')
```

[23]:
```
'toto'
```

[24]:
```
# Situation la plus courante d'utilisation de lambda:
# définir une fonction sans la nommer

def applique_sur_42(fonction):
    "renvoie fonction(42)"
    return fonction(42)
```

[25]:
```
applique_sur_42(lambda x: x / 7)
```

[25]:
```
6.0
```

[26]:
```
applique_sur_42(lambda x: x * '^')
```

[26]:
```
'^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^'
```

3.5.9 Exercices

3.6 Utilisation avancée des conteneurs

3.6.1 Un conteneur de plus : `set`

Un set est non ordonné et ne contient qu'une seule occurrence de chacun de ses éléments (non duplication).

[1]:
```
# construction littéral
{1, 2, 'toto', 2, 3.14}
```

[1]:
```
{1, 2, 3.14, 'toto'}
```

Les opérations d'ensembles sont disponibles

[2]:
```
# union
{2, 3, 4} | {1, 2}
```

[2]:
```
{1, 2, 3, 4}
```

[3]:
```
# intersection
{2, 3, 4} & {1, 2}
```

[3]:
```
{2}
```

[4]:
```
# différence
{2, 3, 4} - {1, 2}
```

[4]:
```
{3, 4}
```

[5]:
```
# différence symétrique
{2, 3, 4} ^ {1, 2}
```

[5]:
```
{1, 3, 4}
```

[6]:
```
# une manière d'enlever les doublons d'une liste
l = [1, 2, 3, 2, 4, 1, 4]
set(l)
```

[6]:
```
{1, 2, 3, 4}
```

3.6.2 Les méthodes des conteneurs de base

[7]:
```
def compare_methodes(type1, type2):
    build_attr = lambda type_: set(e for e in dir(type_) if e[0] != '_')
    attr1 = build_attr(type1)
    attr2 = build_attr(type2)

    print(f'* méthodes communes à {type1.__name__} et {type2.__name__}:')
    print(list(attr1 & attr2))
    print()
    print(f'* méthodes propres à {type1.__name__} :')
    print(list(attr1 - attr2))
    print()
    print(f'* méthodes propres à {type2.__name__} :')
    print(list(attr2 - attr1))
```

list vs tuple

[8]:
```
compare_methodes(list, tuple)
```

```
* méthodes communes à list et tuple:
['count', 'index']
```

(suite sur la page suivante)

(suite de la page précédente)

```
* méthodes propres à list :
['pop', 'append', 'extend', 'sort', 'remove', 'copy', 'reverse', 'insert', 'clear']

* méthodes propres à tuple :
[]
```

[9]:
```
help(list.append)
```

```
Help on method_descriptor:

append(self, object, /)
    Append object to the end of the list.
```

[10]:
```
help(list.extend)
```

```
Help on method_descriptor:

extend(self, iterable, /)
    Extend list by appending elements from the iterable.
```

list vs dict

[11]:
```
compare_methodes(list, dict)
```

```
* méthodes communes à list et dict:
['pop', 'clear', 'copy']

* méthodes propres à list :
['append', 'index', 'extend', 'sort', 'remove', 'count', 'reverse', 'insert']

* méthodes propres à dict :
['values', 'get', 'items', 'keys', 'fromkeys', 'setdefault', 'popitem', 'update']
```

[12]:
```
help(dict.get)
```

```
Help on method_descriptor:

get(self, key, default=None, /)
    Return the value for key if key is in the dictionary, else default.
```

[13]:
```
help(dict.setdefault)
```

```
Help on method_descriptor:

setdefault(self, key, default=None, /)
    Insert key with a value of default if key is not in the dictionary.

    Return the value for key if key is in the dictionary, else default.
```

[14]:
```
help(dict.update)
```

```
Help on method_descriptor:

update(…)
    D.update([E, ]**F) -> None.  Update D from dict/iterable E and F.
    If E is present and has a .keys() method, then does:  for k in E: D[k] = E[k]
    If E is present and lacks a .keys() method, then does:  for k, v in E: D[k] = v
    In either case, this is followed by: for k in F:  D[k] = F[k]
```

list vs set

[15]:
```
compare_methodes(list, set)
```

```
* méthodes communes à list et set:
['clear', 'pop', 'remove', 'copy']

* méthodes propres à list :
['append', 'index', 'extend', 'sort', 'count', 'reverse', 'insert']

* méthodes propres à set :
['difference', 'add', 'union', 'symmetric_difference', 'isdisjoint', 'discard',
↪'intersection', 'intersection_update', 'issubset', 'issuperset', 'symmetric_
↪difference_update', 'difference_update', 'update']
```

[16]:
```
help(set.add)
```

```
Help on method_descriptor:

add(…)
    Add an element to a set.

    This has no effect if the element is already present.
```

dict vs set

[17]:
```
compare_methodes(dict, set)
```

```
* méthodes communes à dict et set:
['pop', 'update', 'clear', 'copy']

* méthodes propres à dict :
['values', 'get', 'items', 'keys', 'fromkeys', 'setdefault', 'popitem']

* méthodes propres à set :
['difference', 'add', 'union', 'symmetric_difference', 'isdisjoint', 'discard',
↪'intersection', 'intersection_update', 'issubset', 'issuperset', 'remove',
↪'symmetric_difference_update', 'difference_update']
```

3.6.3 Exercices

— *Compte occurences (étape 2)*

3.7 Bien boucler en Python

En résumé :
— dict.items(), dict.values()
— sorted, enumerate, zip
— all, any
— min, max
— sum
— map
Lecture : Loop like a native (http://nedbatchelder.com/text/iter/iter.html)

D'autres outils encore dans le module standard `itertools`.

3.7.1 Comment boucler sur un dict ?

[1]:
```
mon_dict = {'un': 1, 'deux': 2, 'trois': 3, 'quatre': 4}
```

[2]:
```
# parcourir les clefs

for clef in mon_dict:
    print('clef :', clef)
```

```
clef : un
clef : deux
clef : trois
clef : quatre
```

[3]:
```
# parcourir les valeurs

for valeur in mon_dict.values():
    print('valeur :', valeur)
```

```
valeur : 1
valeur : 2
valeur : 3
valeur : 4
```

[4]:
```
# parcourir les pairs clef-valeur

for clef, valeur in mon_dict.items():
    print(f'clef : {clef:>10}   |  valeur : {valeur:>10}')
```

```
clef :         un   |  valeur :          1
clef :       deux   |  valeur :          2
clef :      trois   |  valeur :          3
clef :     quatre   |  valeur :          4
```

3.7.2 Changer la manière d'itérer

[5]:
```
# sorted

l = [4., 6., 2., 4., 9., 1.]

for i in sorted(l):
    print(i, end=' ')
```

```
1.0 2.0 4.0 4.0 6.0 9.0
```

[6]:
```
# enumerate

for indice, valeur in enumerate(l):
    print(f'l[{indice:d}] = {valeur:.2f}')
```

```
l[0] = 4.00
l[1] = 6.00
l[2] = 2.00
l[3] = 4.00
l[4] = 9.00
l[5] = 1.00
```

[7]:
```
# zip

def f(a, b):
    return f'a = {a:>8} | b = {b:>8}'

list_a = ['un', 'deux', 'trois', 'quatre']
list_b = [1, 2, 3, 4]

# comment appliquer f sur list_a-list_b ?

for (a, b) in zip(list_a, list_b):
    print(f(a, b))
```

```
a =       un | b =        1
a =     deux | b =        2
a =    trois | b =        3
a =   quatre | b =        4
```

[8]:
```
list(zip(list_a, list_b))
```

[8]:
```
[('un', 1), ('deux', 2), ('trois', 3), ('quatre', 4)]
```

3.7.3 Parfois, il n'y a même pas besoin de faire sa boucle

all / any

[9]:
```
# on pourrait avoir besoin de ceci

def tous_vrai(iterable):
    for element in iterable:
        if not element:
```

(suite sur la page suivante)

(suite de la page précédente)

```
            return False
    return True

# mais cela existe déjà

help(all)
```

```
Help on built-in function all in module builtins:

all(iterable, /)
    Return True if bool(x) is True for all values x in the iterable.

    If the iterable is empty, return True.
```

[10]:
```
# on pourrait avoir besoin de ceci

def au_moins_un_vrai(iterable):
    for element in iterable:
        if element:
            return True
    return False

# mais cela existe déjà

help(any)
```

```
Help on built-in function any in module builtins:

any(iterable, /)
    Return True if bool(x) is True for any x in the iterable.

    If the iterable is empty, return False.
```

max / min

[11]:
```
# on pourrait avoir besoin de ceci

def max_sequence(iterable):
    copie = list(iterable)
    result = copie[0]
    for valeur in copie[1:]:
        result = max(result, valeur)
    return resulr

# mais cela existe déjà

help(max)
```

```
Help on built-in function max in module builtins:

max(…)
```

(suite sur la page suivante)

(suite de la page précédente)

```
    max(iterable, *[, default=obj, key=func]) -> value
    max(arg1, arg2, *args, *[, key=func]) -> value

    With a single iterable argument, return its biggest item. The
    default keyword-only argument specifies an object to return if
    the provided iterable is empty.
    With two or more arguments, return the largest argument.
```

sum

[12]:
```
# on pourrait avoir besoin de ceci

def somme_sequence(iterable, start=0):
    result = start
    for valeur in iterable:
        result += valeur
    return result

# mais cela existe déjà

help(sum)
```

```
Help on built-in function sum in module builtins:

sum(iterable, /, start=0)
    Return the sum of a 'start' value (default: 0) plus an iterable of numbers

    When the iterable is empty, return the start value.
    This function is intended specifically for use with numeric values and may
    reject non-numeric types.
```

map

[13]:
```
# on pourrait avoir besoin de ceci

def applique_sur_sequence(fontions, iterable):
    result = []
    for valeur in iterable:
        result.append(fonction(valeur))
    return result

# mais cela existe déjà

help(map)
```

```
Help on class map in module builtins:

class map(object)
 |  map(func, *iterables) --> map object
 |
```

(suite sur la page suivante)

(suite de la page précédente)

```
 |  Make an iterator that computes the function using arguments from
 |  each of the iterables.  Stops when the shortest iterable is exhausted.
 |
 |  Methods defined here:
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __next__(self, /)
 |      Implement next(self).
 |
 |  __reduce__(…)
 |      Return state information for pickling.
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
```

3.7.4 Exercices

Creer un dict à partir d'une liste de clefs et d'une liste de valeurs.

Etant donné un dict dont les valeurs sont des entiers, extraire le sous-dict composé uniquement des valeurs paires.

3.8 Lire et écrire des fichiers

Python peut lire et écrire des fichiers, *via* la commande `open()`.

On se limitera aux fichiers-texte, bien que Python supporte le format binaire.

[1]:
```
# Ouverture du fichier
fichier = open('open-files_exo1.txt', 'r')

# Extraction des données d'un seul bloc
data = fichier.read()

# Fermeture du fichier
fichier.close()

# Et maintenant, tous ensemble :
print(data)
```

```
Young man, there's no need to feel down.
I said, young man, pick yourself off the ground.
I said, young man, 'cause you're in a new town
There's no need to be unhappy.
```

(suite sur la page suivante)

(suite de la page précédente)

```
Young man, there's a place you can go.
I said, young man, when you're short on your dough.
You can stay there, and I'm sure you will find
Many ways to have a good time.

It's fun to stay at the y-m-c-a.
It's fun to stay at the y-m-c-a.
```

Note : On préférera l'utilisation du *context manager* `with`.

[2]:
```
# Le fichier est ouvert automatiquement à l'entrée du with

with open('open-files_exo1.txt', 'r') as fichier:
    data = fichier.readlines()  # Extraction des données en lignes
    print('fichier est fermé (dans le with) ?', fichier.closed)

# Le fichier est fermé automatiquement à la sortie du with
print('fichier est fermé ?', fichier.closed)

data
```

```
fichier est fermé (dans le with) ? False
fichier est fermé ? True
```

[2]:
```
["Young man, there's no need to feel down. \n",
 'I said, young man, pick yourself off the ground. \n',
 "I said, young man, 'cause you're in a new town \n",
 "There's no need to be unhappy. \n",
 '\n',
 "Young man, there's a place you can go. \n",
 "I said, young man, when you're short on your dough. \n",
 "You can stay there, and I'm sure you will find \n",
 'Many ways to have a good time. \n',
 '\n',
 "It's fun to stay at the y-m-c-a. \n",
 "It's fun to stay at the y-m-c-a. \n"]
```

[3]:
```
# Le mode 'w' donne le mode en écriture
with open('open-files_exo1_copie.txt','w') as fichier :
    fichier.writelines(data)

# Append, 'a', permet de compléter le fichier sans le remplacer
with open('open-files_exo1_copie.txt','a') as fichier :
    fichier.writelines(data)
```

[4]:
```
# `fichier` est itérable, chaque itération renvoie une ligne du fichier

with open('open-files_exo1.txt') as fichier :
    for ligne in fichier:
        print('**', ligne, end='')  # <- end='' évite l'ajout d'un saut de ligne␣
↪supplémentaire
```

```
** Young man, there's no need to feel down.
** I said, young man, pick yourself off the ground.
** I said, young man, 'cause you're in a new town
** There's no need to be unhappy.
**
** Young man, there's a place you can go.
** I said, young man, when you're short on your dough.
** You can stay there, and I'm sure you will find
** Many ways to have a good time.
**
** It's fun to stay at the y-m-c-a.
** It's fun to stay at the y-m-c-a.
```

[5]:
```
# attention !
# `ligne` contient le symbole de fin de ligne

repr(ligne)
```

[5]:
```
'"It\'s fun to stay at the y-m-c-a. \\n"'
```

3.8.1 Exercices

— *Statistiques d'un échantillon*

3.9 Aperçu des fonctions *built-in*

https://docs.python.org/library/functions.html

3.9.1 Exercices

3.10 Noms et objets non immuables

Nous utilisons ici http://pythontutor.com pour visualiser pas à pas le déroulement d'un programme python et l'évolution des variables.

3.10.1 Un nom est une référence, pas une variable ...

— … la variable est l'objet référencé ; …
— … et "=" ne fait donc pas de copie de variable.

En Python, il est plus juste de dire que "=" *relie un nom à un objet*.

3.10.2 Malgré tout ça

— on ne se gênera pas pour parler de variables
— on se contentera d'être pragmatique
— la surprise est de courte durée

3.10.3 Deux noms peuvent désigner la même chose...

[1]:
```
a = [5]
b = a
b[0] = 3

print(f"{a = }\n{b = }")
```

```
a = [3]
b = [3]
```

[2]:
```
from IPython.display import IFrame
IFrame("https://pythontutor.com/iframe-embed.html#code=a+%3D+%5B5%5D%0Ab+%3D+a%0Ab%5B0
↪%5D+%3D+3&origin=opt-frontend.js&cumulative=false&heapPrimitives=true&
↪drawParentPointers=false&textReferences=false&showOnlyOutputs=false&py=3&
↪rawInputLstJSON=%5B%5D&curInstr=0&codeDivWidth=350&codeDivHeight=400",
       width='100%', height=300)
```

[2]:
```
<IPython.lib.display.IFrame at 0x1f1ce64e920>
```

lien (http://pythontutor.com/visualize.html#code=a+%3D+%5B5%5D%0Ab+%3D+a%0Ab%5B0%5D+%3D+3&origin=opt-frontend.js&cumulative=false&heapPrimitives=true&drawParentPointers=false&textReferences=false&showOnlyOutputs=false&py=3&rawInputLstJSON=%5B%5D&curInstr=0)

3.10.4 ... mais pas toujours

[3]:
```
# Deux noms PEUVENT désigner la même chose
# mais pas pour toujours

i = 5
j = i
j = 3

print(f"{i = }\n{j = }")
```

```
i = 5
j = 3
```

[4]:
```
from IPython.display import IFrame
IFrame("https://pythontutor.com/iframe-embed.html#code=i+%3D+5%0Aj+%3D+i%0Aj+%3D+3&
↪origin=opt-frontend.js&cumulative=false&heapPrimitives=true&
↪drawParentPointers=false&textReferences=false&showOnlyOutputs=false&py=3&
↪rawInputLstJSON=%5B%5D&curInstr=0&codeDivWidth=350&codeDivHeight=400",
       width='100%', height=300)
```

[4]:
```
<IPython.lib.display.IFrame at 0x1f1ce64e800>
```

lien (http://pythontutor.com/visualize.html#code=i+%3D+5%0Aj+%3D+i%0Aj+%3D+3&origin=opt-frontend.js&cumulative=false&heapPrimitives=true&drawParentPointers=false&textReferences=false&showOnlyOutputs=false&py=3&rawInputLstJSON=%5B%5D&curInstr=0)

3.10.5 Attention aux arguments de type “list”

[5]:
```
# Fonction test

def somme(x) :
    "Somme x avec lui-même"
    x += x
    return x
```

[6]:
```
# Test sur un entier

a = 2

print('avant exécution : ', f"{a = }")

b = somme(a)

print('après exécution : ', f"{a = }")
print('après exécution : ', f"{b = }")
```

```
avant exécution :  a = 2
après exécution :  a = 2
après exécution :  b = 4
```

[7]:
```
# Test sur une liste

a = [1, 2]

print('avant exécution : ', f"{a = }")

b = somme(a)

print('après exécution : ', f"{a = }")
print('après exécution : ', f"{b = }")
```

```
avant exécution :  a = [1, 2]
après exécution :  a = [1, 2, 1, 2]
après exécution :  b = [1, 2, 1, 2]
```

[8]:
```
# Mais que se passe-t-il?
from IPython.display import IFrame
IFrame("https://pythontutor.com/iframe-embed.html#code=def+somme(x)+%3A%0A++++
↪%23+On+somme+x+avec+lui-m%C3%AAme%0A++++x+%2B%3D+x%0A++++return+x%0A%0A%23+entier
↪%0Aa+%3D+2%0Ab+%3D+somme(a)%0A%0A%23+liste%0Aa+%3D+%5B1,+2%5D%0Ab+%3D+somme(a)&
↪origin=opt-frontend.js&cumulative=false&heapPrimitives=true&
↪drawParentPointers=false&textReferences=false&showOnlyOutputs=false&py=3&
↪rawInputLstJSON=%5B%5D&curInstr=0&codeDivWidth=350&codeDivHeight=400",
      width='100%', height=500)
```

[8]:
```
<IPython.lib.display.IFrame at 0x1f1ce64c6d0>
```

lien (http://pythontutor.com/visualize.html#code=def+somme(x)+%3A%0A++++%23+On+somme+x+avec+lui-m%C3%AAme%0A++++x+%2B%3D+x%0A++++return+x%0A%0A%23+entier%0Aa+%3D+2%0Ab+%3D+somme(a)%0A%0A%23+liste%0Aa+%3D+%5B1,+2%5D%0Ab+%3D+somme(a)&origin=opt-frontend.js&cumulative=false&heapPrimitives=true&drawParentPointers=false&textReferences=false&showOnlyOutputs=

false&py=3&rawInputLstJSON=%5B%5D&curInstr=0)

3.10.6 Portée des variables

La résolution d'un nom suit la règle dite *LEGB* (Local > Enclosed > Global > Built-in).

Assigner un nom *localement* (i.e. au sein d'une fonction) ne modifie pas le niveau global.

Le mot clef ``**global**`` permet de modifier cette règle.

[9]:
```
# pas de danger à utiliser le même nom localement et globalement

X = 'toto'

def f():
    X = 42
    print(f"local : {X = }")

f()
print(f"global: {X = }")
```

```
local : X = 42
global: X = 'toto'
```

[10]:
```
# Un cas fréquent (et potentiellement dangereux) est l'utilisation d'une variable␣
↪globale dans une fonction
X = 3
def triple(a):
    return a*X

triple(3)
```

[10]:
```
9
```

[11]:
```
# Catastrophe : la fonction a changé de valeur retournée.
X = 4
triple(3)
```

[11]:
```
12
```

conclusion : il faut assumer ses responsabilités quand on *modifie* un objet.

remarque : si le nom est une référence, il n'y a pas de pointeur natif en python.

3.10.7 Exercices

Explorer la liste `liste_de_liste` construite par :

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

Que se passe-t-il quand on modifie un élément de x ? de y ?

Plus de fun ! Construire une liste *récursive* (dont un élément est elle-même ou bien la contient).

— *Matrice de zéros*

3.11 Gérer les exceptions

3.11.1 Qu'est-ce donc ?

Une exception est levée quand une erreur est commise

[1]:
```
nexistepas
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[1], line 1
----> 1 nexistepas

NameError: name 'nexistepas' is not defined
```

[2]:
```
2 / 0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Cell In[2], line 1
----> 1 2 / 0

ZeroDivisionError: division by zero
```

[3]:
```
'un' + 2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[3], line 1
----> 1 'un' + 2

TypeError: can only concatenate str (not "int") to str
```

[4]:
```
'bonjour'[9]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[4], line 1
----> 1 'bonjour'[9]

IndexError: string index out of range
```

[5]:
```
{'un': 1}['deux']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[5], line 1
----> 1 {'un': 1}['deux']

KeyError: 'deux'
```

Le *Traceback* permet de suivre l'imbrication des appels au moment de l'erreur

[6]:
```
def f0(x):
    return x / 0
```

(suite sur la page suivante)

(suite de la page précédente)

```
def f1(x):
    return f0(x) / 1

def f2(x):
    return f1(x) / 2

def f3(x):
    return f2(x) / 3

f3(1)
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Cell In[6], line 13
     10 def f3(x):
     11     return f2(x) / 3
---> 13 f3(1)

Cell In[6], line 11, in f3(x)
     10 def f3(x):
---> 11     return f2(x) / 3

Cell In[6], line 8, in f2(x)
      7 def f2(x):
----> 8     return f1(x) / 2

Cell In[6], line 5, in f1(x)
      4 def f1(x):
----> 5     return f0(x) / 1

Cell In[6], line 2, in f0(x)
      1 def f0(x):
----> 2     return x / 0

ZeroDivisionError: division by zero
```

Il est possible d'émettre soit même une exception pour signaler une erreur.

[7]:
```
def fonction_qui_n_aime_pas_toto(valeur):
    """ affiche valeur
    Je n'aime pas Toto, donc 'toto' est une valeur interdite !
    """
    if valeur == 'toto':
        raise ValueError("T'es pas beau, toto !")  # pensez à mettre un message utile.
↪..
    print(valeur)
```

[8]:
```
fonction_qui_n_aime_pas_toto('tata')
```

```
tata
```

```
[9]: fonction_qui_n_aime_pas_toto('toto')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[9], line 1
----> 1 fonction_qui_n_aime_pas_toto('toto')

Cell In[7], line 6, in fonction_qui_n_aime_pas_toto(valeur)
      2 """ affiche valeur
      3 Je n'aime pas Toto, donc 'toto' est une valeur interdite !
      4 """
      5 if valeur == 'toto':
----> 6     raise ValueError("T'es pas beau, toto !")  # pensez à mettre un message␣
↪utile…
      7 print(valeur)

ValueError: T'es pas beau, toto !
```

3.11.2 De nombreuses exceptions sont définies de base

https://docs.python.org/2/library/exceptions.html#exception-hierarchy

3.11.3 try / except / else /finally

Il est possible de "sécuriser" une portion de code. Voire même d'intégrer l'erreur dans la logique du code.

```
try:
    # bloc d'instruction dont on veux contrôler les exceptions émises
except ExceptionType1 as err1:
    # Traitement de err1
    # Typiquement:
    #    - on a une solution de repli dans le cas de err1
    #    OU
    #    - on sauve les meubles et on fait rebondir err1 (ou une nouvelle erreur)
except ExceptionType2 as err2:
    # Traitement de err2
    # ...
except ExceptionType3 as err3:
    # Traitement de err3
    # ...
else:
    # Si aucune exception n'est émise dans le try
    # alors ce bloc est exécuté
finally:
    # Ce bloc sera exécuté quoi qu'il arrive
```

3.11.4 Exercices

Ecrire une fonction `get(obj, index, default)` avec le même comportement que `dict.get()` mais qui marche aussi bien pour les dict ou les listes .

Ecrire une fonction `map_with_default(function, sequence, default)` qui marche comme `map(function, sequence)` sauf que si une erreur est produite par `function` lors de son évaluation, l'erreur n'est pas propagée. A la place, la valeur manquante est assignée à `default`.

3.12 Listes en intension

Où comment faire des boucles sans en faire (et sans `map` et `filter`)

[1]:
```
# list comprehension

iterable = range(10)

[i ** 2 for i in iterable if (i % 2 == 0)]
```

[1]:
```
[0, 4, 16, 36, 64]
```

[2]:
```
# dict comprehension

iterable = ['toto', 'tata', 'titi']

{nom: num for (num, nom) in enumerate(iterable)}
```

[2]:
```
{'toto': 0, 'tata': 1, 'titi': 2}
```

[3]:
```
# generator comprehension

iterable = ['bonjour', None, None, 'tout', None, 'le', None, None, None, 'monde']

generator = (mot for mot in iterable if mot is not None)

print(generator)
' '.join(generator)
```

```
<generator object <genexpr> at 0x00000239D2686180>
```

[3]:
```
'bonjour tout le monde'
```

3.12.1 Exercices

— *Filtrage de liste*

3.13 Fonctions d'ordre supérieur

On parle de fonction d'ordre supérieur quand cette fonction :

— prend une ou plusieurs fonctions en argument
— renvoie une fonction

Chacune de ces deux propriétés est possible en Python. Il s'agit souvent d'un moyen rendre un code plus modulaire en séparant les logiques internes.

[1]:
```
# Un exemple de fonctions que l'on peut composer entre elles


def pain(func):
    def wrapper():
        print("</''''''''''\>")
        func()
        print("<\__________/>")
    return wrapper
```

(suite sur la page suivante)

(suite de la page précédente)

```
def garniture(func):
    def wrapper():
        print(" OO tomate OO ")
        func()
        print(" ~~ salade ~~ ")
    return wrapper


def jambon():
    print(" -- jambon -- ")


sandwich = pain(garniture(jambon))

print("Bonne appétit")
print()
sandwich()
```

```
Bonne appétit

</''''''''''\>
 OO tomate OO
 -- jambon --
 ~~ salade ~~
<\__________/>
```

3.13.1 Exercices

Ecrire le corps de la fonction suivante :

```
def deriv(func, epsilon=1e-6):
    """ renvoie la fonction dérivée de func

    Le calcul de la dérivée est fait par différence finie:

        f_prime(x) = ( f(x+epsilon) - f(x) ) / epsilon

    """
    pass
```

3.14 Les générateurs

Les générateurs sont :

— des fabriques à itérables

— sont créés comme des fonctions mais utilisent `yield` à la place de `return`.

Ils s'agit donc d'un moyen simple de réaliser des boucles complexes.

[1]:
```
# Principe de mise en oeuvre
```

(suite sur la page suivante)

(suite de la page précédente)

```
def carre_decroissant(valeur):  # def comme pour une fonction
    valeur = int(valeur)
    while valeur:               # une boucle, ici : while
        yield valeur ** 2       # yield dans la boucle
        valeur -= 1

print(type(carre_decroissant))
print(carre_decroissant)
```

```
<class 'function'>
<function carre_decroissant at 0x7fb1e02c7d90>
```

[2]:
```
# construction d'un générateur

generateur = carre_decroissant(5)

print(type(generateur))
print(generateur)
```

```
<class 'generator'>
<generator object carre_decroissant at 0x7fb1e02abdb0>
```

[3]:
```
# exploitation du générateur

for valeur in generateur:
    print(valeur)
```

```
25
16
9
4
1
```

3.14.1 Exercices

— *Suite de Collatz (bonus)*

3.15 Rendre un script importable

Voici le contenu typique d'un script python

```
## fichier: mon_script_bof.py

...  # on fait des trucs compliqués
print("message important")
```

Le problème avec « mon_script_bof.py » : on ne peut pas faire `import mon_script_bof` sans lancer effectivement le calcul.

La solution est dans la formule magique : `if __name__ == '__main__': ...`. Ce qui suit le `if` n'est pas exécuter lors de `import`.

```
## fichier: mon_script_bien.py

def main():
    ...  # on fait des trucs compliqués
    print("message important")


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

4 Au-delà du langage : les modules

4.1 Qu'est-ce qu'un module ?

Un module est un ensemble de fonctions et classes dédiées à une tâche précise rangées sous un même nom.

4.1.1 Utilisation

[1]:
```
# Pour importer un module, on utilise la command `import`
import math
type(math)
```

[1]:
```
module
```

[2]:
```
# Tout élément du module est accessible via le `.`
math.pi
```

[2]:
```
3.141592653589793
```

[3]:
```
math.e
```

[3]:
```
2.718281828459045
```

[4]:
```
math.sqrt(2)
```

[4]:
```
1.4142135623730951
```

On peut n'importer que certains éléments du module

[5]:
```
# e n'est pas défini pour le moment, seulement math.e
e
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[5], line 2
      1 # e n'est pas défini pour le moment, seulement math.e
----> 2 e

NameError: name 'e' is not defined
```

[6]:
```
from math import cos, e, pi
```

[7]:
```
# désormais e est défini, même valeur que math.e
e
```

[7]:
```
2.718281828459045
```

[8]:
```
cos(pi)
```

[8]:
```
-1.0
```

Mais on préfèrera l'utilisation du `.` pour une meilleure traçabilité.

4.1.2 Les *Standard Libraries*

Python dispose de nombreuses librairies par défaut, qui couvrent une grande variété de besoins.

4.1.3 Les modules tierces

En plus des modules standards, on trouve les *modules tiers* dont une grande part est offerte (https://fr.wikipedia.org/wiki/Free/Libre_Open_Source_Software) par la communauté.

4.1.4 Créer un module

Un module simple est un fichier Python normal. On peut donc créer facilement ses propres modules. Il suffit de le placer au bon endroit.

L'instruction

```
import monbomodule
```

cherche le fichier `monbomodule.py` à 2 endroits :

1. dans le répertoire où se trouve le script qui fait l'import
2. dans les répertoires où sont **installés** les modules standards et tierce

Il suffit de mettre son module au même endroit que sont script.

Note : package

Si vous travaillez trop fort, vous risquez d'avoir trop de modules. Dans ce cas, il faut créer un package (https://docs.python.org/2/tutorial/modules.html#packages), ce qui consiste principalement mettre les modules dans un répertoire.

4.1.5 Pourquoi créer un module ?

- Pour structurer son code
  - ranger les fonctions qui marchent ensemble au même endroit
  - séparer les choses qui n'ont rien à voir entre elles
  - => un code plus facile à lire et augmenter
- Pour réutiliser son code
  - debugger une seule fonction pour corriger tous les scripts qui s'en servent = souffrance / 10
  - améliorer une seule fonction pour accélérer tous les scripts qui s'en servent = bonheur * 10
  - => karma * 100

4.1.6 Exercices

Mon premier module : rédiger un module qui comprend la fonction `double`, et l'utiliser dans Python

4.2 *String Services*

Ensemble de modules utiles pour la manipulation de chaînes de caractères.

4.2.1 Exemple des *regular expressions*

Les *regular expressions* (ou *regexp*) sont des chaînes de caractères codifiées décrivant des ensembles de chaînes de caractères « de base ».

Elles permettent de retrouver rapidement un ensemble de caractères dans une chaîne.

[1]:
```
# On importe le module
import re
```

[2]:
```
# On cherche un flottant dans une chaine de caractères
chaine = 'La valeur de pi à 10 décimales près est 3.1415926536. Très intéressant, non?
↪'
re_float = '\d+.\d+'
valeur_pi = re.search(re_float,chaine)
valeur_pi.group()
```

[2]:
```
'3.1415926536'
```

[3]:
```
# On extrait tous les mots d'une phrase
chaine = 'Mais qui+ a::mis -le%$¨bazard!!dans~~cette789chaine?'
re_word = '[a-zA-Z\?]+'
mots = re.findall(re_word,chaine)
mots
```

[3]:
```
['Mais', 'qui', 'a', 'mis', 'le', 'bazard', 'dans', 'cette', 'chaine?']
```

[4]:
```
# On peut remplacer une séquence par une autre
chaine = 'Farid est vraiment mon animateur préféré.'
print(chaine)
chaine = re.sub('Farid','Théophile',chaine)
print(chaine)
```

```
Farid est vraiment mon animateur préféré.
Théophile est vraiment mon animateur préféré.
```

Les *regexp* prennent tout leur intérêt dans le traitement de grandes chaînes de caractères.

Elles sont notamment utiles pour accéder rapidement à une donnée précise dans un gros fichier.

4.2.2 Exercices

4.3 *Data Types*

Modules introduisant des types de données spécifiques.

4.3.1 Module *collections*

collections proposent 5 nouveaux conteneurs.

```
import collections

collections.namedtuple   # factory function for creating tuple subclasses with named␣
↪fields
collections.deque        # list-like container with fast appends and pops on either␣
↪end
collections.Counter      # dict subclass for counting hashable objects
collections.OrderedDict  # dict subclass that remembers the order entries were added
collections.defaultdict  # dict subclass that calls a factory function to supply␣
↪missing values
```

4.3.2 Exercices

Aidé du module `collections`, réécrire une fonction `compte_occurences()`.

4.4 *Numeric tools*

Ensemble de modules utiles au traitement numérique et mathématique

4.4.1 Module *math*

Comprend les fonctions mathématiques de base, et 2 flottants : *pi* et *e*

[1]:
```
import math
# Exponentielle
print(math.exp(5))
print(math.exp(5) == math.exp(2)*math.exp(3))
```

```
148.4131591025766
True
```

[2]:
```
# Log
print(math.log(10))
print(math.log(10,10))
```

```
2.302585092994046
1.0
```

[3]:
```
# Trigonométrie
t=math.tan(math.pi/6)
print(t)
print(math.atan(t) == math.pi/6)
```

```
0.5773502691896257
True
```

4.4.2 Module *cmath*

Etant le domaine de validité des fonctions du module *math* aux nombres complexes

[4]:
```
import cmath
```

[5]:
```
# Python admet un type de données 'complex'
complex1 = complex(0.,1.)
print(complex1)
type(complex1)
```

```
1j
```

[5]:
```
complex
```

[6]:
```
# Un entier/flottant accompagné de la lettre 'j' est automatiquement interprété comme␣
↪un nombre complexe
complex2 = -5+18j
type(complex2)
```

[6]:
```
complex
```

[7]:
```
# Les parties réelle et imaginaire sont accessibles séparément
print(complex2.real)
print(complex2.imag)
```

```
-5.0
18.0
```

[8]:
```
# L'arithmétique de base est prise en charge nativement
print(complex1 + complex2)
print(complex1 * complex2)
```

```
(-5+19j)
(-18-5j)
```

[9]:
```
# cmath permet de traiter les onctions complexes
print(cmath.exp(complex1))
print(cmath.log(complex2))
```

```
(0.5403023058681398+0.8414709848078965j)
(2.9275359611012135+1.8417431771333173j)
```

4.4.3 Module *random*

Traite les nombres aléatoires

[10]:
```
import random
```

[11]:
```
# Génère des flottants selon une certaines distribution statistique
uniforme = [random.uniform(0,1) for i in range(5)]
normale = [random.gauss(0,1) for i in range(5)]
print(uniforme)
print(normale)
```

```
[0.13484883369952094, 0.9020293810800457, 0.7848925714114228, 0.19127230774902915, 0.
↪8887251865365928]
[-0.9301816284842389, 0.422877012978363, 0.8100489451767664, 0.512158683146731, -1.
↪8614993820726016]
```

[12]:
```
# Traitement de séquences
echantillon = ['Python','C++','Java','Ruby']
print(random.choice(echantillon))
random.shuffle(echantillon)
print(echantillon)
```

```
Python
['Python', 'Ruby', 'C++', 'Java']
```

4.4.4 Exercices

4.5 *File and Directory Access*

Modules facilitant le travail sur une arborescence

4.5.1 Module *glob*

Permet de retrouver des fichiers (même syntaxe que sur Linux)

[1]:
```
import glob
# Retourne tous les dossiers et fichiers du dossier courant
glob.glob('*')
```

[1]:
```
['intro.ipynb',
 'modules-tiers.ipynb',
 'stdlib-data-types.ipynb',
 'stdlib-file-directory.ipynb',
 'stdlib-numeric-tools.ipynb',
 'stdlib-string-services.ipynb',
 'stdlib-subprocess.ipynb',
 'virtual-environments.ipynb']
```

[2]:
```
# Retourne uniquement les fichiers concernant les standard libraries
glob.glob('stdlib*.*')
```

[2]:
```
['stdlib-data-types.ipynb',
 'stdlib-file-directory.ipynb',
 'stdlib-numeric-tools.ipynb',
 'stdlib-string-services.ipynb',
 'stdlib-subprocess.ipynb']
```

4.5.2 Module *os.path*

Aide au parcours d'une arborescence

[3]:
```
import os
# Dossier courant
```

(suite sur la page suivante)

(suite de la page précédente)

```
cwd = os.getcwd()
cwd
```

[3]:
```
'D:\\Documents\\smai\\Travail\\CODES\\formation-python\\notebooks\\python-modules'
```

[4]:
```
# Retourne le dossier parent
print(os.path.dirname(cwd))
# Vérifie l'existence de fichiers
print(os.path.isfile(os.path.join(cwd,'intro.ipynb')))
# Gère les chemins relatifs (mais déconseillé)
print(os.path.isfile('intro.ipynb'))
# Transforme des chemins relatifs en chemins absolus
print(os.path.abspath('.'))
```

```
D:\Documents\smai\Travail\CODES\formation-python\notebooks
True
True
D:\Documents\smai\Travail\CODES\formation-python\notebooks\python-modules
```

4.5.3 Module *shutil*

Gère la création de fichiers

```
import shutil
# Copie de fichiers
shutil.copy(source,destination)
# Copie d'une arborescence
shutil.copytree(source,destination)
# Suppression d'une arborescence
shutil.rmtree(dossier_maitre)
```

4.5.4 Module *pathlib*

os.path + *glob* + *open* dans un seul objet *Path* !

[5]:
```
from pathlib import Path
cwd = Path()
```

[6]:
```
list(cwd.glob('*'))
```

[6]:
```
[WindowsPath('.ipynb_checkpoints'),
 WindowsPath('intro.ipynb'),
 WindowsPath('modules-tiers.ipynb'),
 WindowsPath('stdlib-data-types.ipynb'),
 WindowsPath('stdlib-file-directory.ipynb'),
 WindowsPath('stdlib-numeric-tools.ipynb'),
 WindowsPath('stdlib-string-services.ipynb'),
 WindowsPath('stdlib-subprocess.ipynb'),
 WindowsPath('virtual-environments.ipynb')]
```

[7]:
```
list(cwd.glob('stdlib*.*'))
```

[7]:
```
[WindowsPath('stdlib-data-types.ipynb'),
 WindowsPath('stdlib-file-directory.ipynb'),
 WindowsPath('stdlib-numeric-tools.ipynb'),
 WindowsPath('stdlib-string-services.ipynb'),
 WindowsPath('stdlib-subprocess.ipynb')]
```

[8]:
```
print(Path("rep/file.txt"))
print(Path("rep/file.txt").parent)
```

```
rep\file.txt
rep
```

[9]:
```
print(cwd / 'intro.ipynb')
(cwd / 'intro.ipynb').is_file()
```

```
intro.ipynb
```

[9]:
```
True
```

[10]:
```
myfile = Path('test/toto.txt')
myfile.parent.mkdir(exist_ok=True)
myfile.touch()
myfile.write_text("hello !")
new = myfile.replace('test/titi.txt')
new.read_text()
```

[10]:
```
'hello !'
```

4.5.5 Exercices

4.6 Le module *subprocess*

Python fourmille de librairies standard très pratiques, et une liste exhaustive ne peut être proposée.

Le module *subprocess* peut s'avérer être très utile pour piloter d'autres logiciels.

[ ]:
```
import subprocess as sp
# Lance le bloc-notes
app = sp.Popen('notepad')
print("L'application a été lancée")
```

[ ]:
```
# On peut bloquer le flux d'exécution tant que le logiciel n'a pas terminé
app = sp.Popen('notepad')
app.wait()
print("L'application a été lancée")
```

[ ]:
```
# On peut récupérer les messages de sortie du logiciel
# Exemple avec un script shell qui affiche le contenu du dossier courant
batch = sp.Popen('dir',shell=True,stdout=sp.PIPE)
out,err = batch.communicate()
print(out.decode(errors='replace'))
```

subprocess permet de lancer une batterie de calculs pour le week-end, piloter des échanges d'informations entre des logiciels, …

4.6.1 Exercices

4.7 Les modules tiers

De nombreux modules existent en dehors des *standard libraries*.

Certains utilisateurs développent des modules pour une application particulière, et partagent leur code source.

Le dépôt officiel des modules tiers : https://pypi.python.org/pypi

4.7.1 Installation vs local

Il est très facile de charge un module présent en local mais non installer.

L'interêt de l'installation est qu'il n'y a pas besoin de dupliquer les sources du module pour pour en profiter dans n'importe quel programme que vous écrivez.

4.7.2 L'outil `pip`

`pip` aide à installer et gérer les modules Python.

Il s'utilise directement depuis la ligne de commande système :

```
pip search mot_clef
pip install nom_de_module
pip install --user nom_de_module
pip uninstall nom_de_module
pip help
```

Dans le cas classique, "`pip install`" va interroger le dépôt *PyPI*, télécharger une archive du module puis déployer l'installation.

4.7.3 L'outil `conda`

La distribution Anaconda (https://store.continuum.io/cshop/anaconda/) met à disposition la commande `conda` ainsi qu'un dépôt de paquets (http://repo.continuum.io/pkgs/index.html) à vocation scientifique

`conda` permet de d'installer et de gérér finement ces paquets. Il rend particulièrement simple l'installation de modules nécessitant la compilation de code (C/C++/Fortran).

Le dépôt est moins riche que PyPI mais pensez quand même à conda pour les modules difficiles à installer.

```
conda search mot_clef
conda install nom_de_module
conda remove nom_de_module
conda help
```

4.7.4 Exercices

4.8 Les environnements virtuels

4.8.1 Contexte

Le problème

1. Quand on installe un module, il installe/update les modules dont il a besoin.
2. Les modules ont des numéros de version et donne des contraintes sur les versions de leurs dépendences.

3. Si deux modules ont une dépendence commune, il n'est pas sûr que les contraintes sur les versions soient compatibles.
4. Plus on installe de modules, plus les conflits sont probables.

— On peut accepter de revenir sur une version antérieur pour obtenir la compatibilité.
  — (pas toujours possible)
— Mais pourquoi être contraint sur la version d'un module A à cause d'un module B alors que A et B ne seront jamais utilisés ensemble ?

Il ne faut pas installer simultanément les modules A et B ! Comment faire ?

La solution

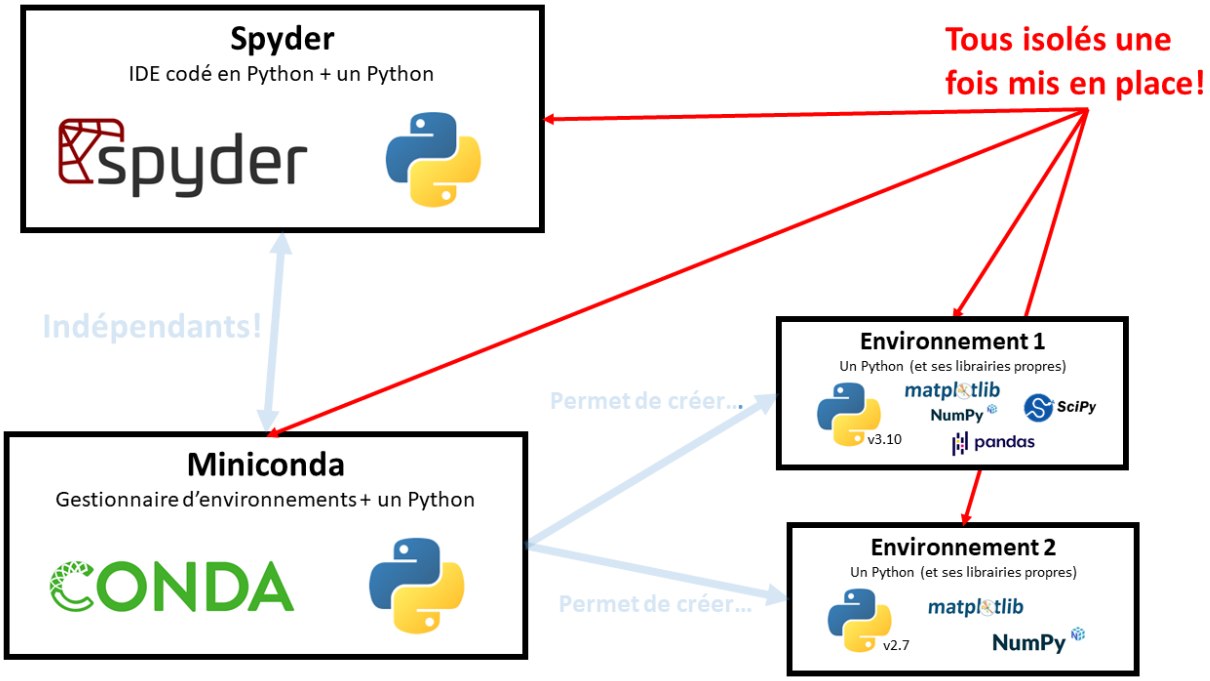
Faire des boîtes !

Environnement virtuel (cf doc (https://docs.python.org/fr/3/glossary.html#term-virtual-environment)) :

> Environnement d'exécution isolé (en mode coopératif) qui permet aux utilisateurs de Python et aux applications d'installer et de mettre à jour des paquets sans interférer avec d'autres applications Python fonctionnant sur le même système.

Il y a plusieurs manières d'y arriver :
— une solution standard : venv (https://docs.python.org/fr/3/tutorial/venv.html)
— une solution très employée avec python scientifique : conda environments (https://docs.conda.io/projects/conda/en/latest/user-guide/concepts/environments.html)

Dans les deux cas, il s'agit d'outils en ligne de commande qui proposent les actions essentielles suivantes :
— créer un environnement virtuel
  — à faire une fois par environnement
— activer un environnement
  — à faire dans chaque console où l'on souhaite travailler
  — l'environnement actif est indiqué dans la console
— une fois l'environnement activé, les installations se font dans l'environnement



4.8.2 Conda environments

Documentation complète (https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html)

`conda` est un outil en ligne de commande, en plus d'installer des modules, peut créer des environnements virtuels complètement isolés.

En bref

Créer l'environnement `toto` avec python 3.10

```
conda create -n toto python=3.10
```

Activer l'environnement `toto`

```
conda activate toto
```

Désactiver l'environnement courant

```
conda deactivate
```

Installer des modules dans l'environnement **actif**

```
pip install truc
conda install truc
```

Voir tout ce qui est installé dans un environnement

```
conda list  # environnement actif
conda list -n toto
```

4.8.3 Venv

Documentation complète (https://docs.python.org/3/library/venv.html)

Il s'agit de l'outil standard, fournit directement avec python.

- léger et rapide
- il ne permet pas de gérer plusieurs versions de python simultanément
- il n'est pas compatible avec `conda install`

4.8.4 Exercices

1. Créer un environnement virtuel conda avec python et numpy.
2. Faire le lien entre votre IDE (spyder ?) et cet environnement.

5 Modules scientifiques

5.1 Aperçu de l'écosystème

Trois ingrédients principaux

- Numpy : implémentation performante de tableaux uniformes multi-dimensionnels
- Scipy : implémentation de nombreux algorithmes scientifiques
- Matplotlib : visualisation 2D/3D basée sur le tableaux numpy

Mais aussi

- Pandas : analyse et manipulation de données « fast, powerful, flexible and easy to use »

Et beaucoup d'autres plus spécifiques…

5.1.1 Numpy

http://docs.scipy.org/doc/numpy/reference/
http://docs.scipy.org/doc/numpy/user/index.html
http://wiki.scipy.org/Tentative_NumPy_Tutorial
http://wiki.scipy.org/NumPy_for_Matlab_Users

5.1.2 Scipy

http://docs.scipy.org/doc/scipy/reference/
http://docs.scipy.org/doc/scipy/reference/tutorial/
http://scipy-lectures.github.io/ (parle aussi de Numpy et Matplotlib)

5.1.3 Matplotlib

http://matplotlib.org/
http://www.loria.fr/~rougier/teaching/matplotlib/

5.1.4 Pandas

https://pandas.pydata.org/
https://pandas.pydata.org/docs/getting_started/
https://pandas.pydata.org/docs/user_guide/

5.2 Numpy

5.2.1 La structure de base : le *array*

La contribution majeure de Numpy est de proposer une implémentation performante de tableaux uniformes multi-dimensionnels : le `array`

[1]:
```
# on importe le package numpy.
# il est très fréquent d'abréger son nom en 'np'
import numpy as np
```

[2]:
```
# Le array est un conteneur qui peut être initialisé
# avec une liste, une liste de listes, une liste de listes de listes, ...
# le niveau d'imbrication décrit le nombre de dimensions du array.
x = np.array([[1, 0.0, 3], [0, 1, 5]])
x
```

[2]:
```
array([[1., 0., 3.],
       [0., 1., 5.]])
```

[3]:
```
# 'ndim' est le nombre de dimensions du array
x.ndim
```

[3]:
```
2
```

[4]:
```
# 'shape' informe sur la taille de chaque dimension
# Dans l'exemple, x contient 2 listes à 3 éléments.
x.shape
```

[4]:
```
(2, 3)
```

[5]:
```
# Contrairement aux conteneurs 'classiques', tous les éléments d'un array doivent␣
↪être du même type.
# Dans l'exemple, des flottants.
x.dtype
```

[5]:
```
dtype('float64')
```

[6]:
```
# Numpy dispose de types pour gérer des valeurs non-numériques spécifiques : "Not A␣
↪Number", et "Infinity".
np.NAN, np.Inf
```

[6]:
```
(nan, inf)
```

[7]:
```
# Ces types peuvent cohabiter avec des valeurs numériques
y = np.array([np.NaN, 2], dtype=float)
print(y.dtype)
y
```

```
float64
```

[7]:
```
array([nan,  2.])
```

[8]:
```
# Des fonctions existent pour créer des array aux remplissages particuliers.
# Un array de 0
np.zeros((2, 3), dtype=int)
```

[8]:
```
array([[0, 0, 0],
       [0, 0, 0]])
```

[9]:
```
# Un array de 1
np.ones(5)
```

[9]:
```
array([1., 1., 1., 1., 1.])
```

[10]:
```
# Un array avec un contenu non prédéfini, à remplir par la suite
# (le contenu initial du array sera conditionné par ce qu'il y a en mémoire, mais n
↪'épiloguons pas sur le sujet)
np.empty((5,4), dtype=int)
```

[10]:
```
array([[-1, -1,  0,  0],
       [ 0,  0,  0,  0],
       [ 0,  0,  0,  0],
       [ 0,  0,  0,  0],
       [ 0,  0,  0,  0]])
```

[11]:
```
# Au delà des array, numpy dispose de plusieurs fonctions pratiques
# L'équivalent du range() de Python, mais qui retourne un array
np.arange(6)
```

[11]:
```
array([0, 1, 2, 3, 4, 5])
```

[12]:
```
# Le pendant de np.arange, pour lequel on ne précise pas le pas mais le nombre de␣
↪valeurs
np.linspace(0,6,5)
```

[12]:
```
array([0. , 1.5, 3. , 4.5, 6. ])
```

[13]:
```
# Et bien plus encore...
```

5.2.2 Indexation

L'accès aux éléments d'un `array` est plus souple que dans le cas des conteneurs de base.

[14]:
```
# On créer un array d'entiers
x = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
print(x.dtype)
x
```

```
int32
```

[14]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

[15]:
```
# Le premier indice permet d'accéder aux lignes, ...
x[0]
```

[15]:
```
array([1, 2, 3, 4])
```

[16]:
```
# ... et le deuxième indice aux colonnes (etc pour les array de dimensions␣
↪supérieures)
x[0][2] # marche mais peut mieux faire
```

[16]:
```
3
```

[17]:
```
# ... et le deuxième indice aux colonnes (etc pour les array de dimensions␣
↪supérieures)
x[0, 2] # voilà, là c'est plus propre
```

[17]:
```
3
```

[18]:
```
# On peut accéder aux colonnes en utilisant un slice sur le première indice.
x[:, 0]
```

[18]:
```
array([1, 5, 9])
```

[19]:
```
# Le contenu d'un array peut être modifié
x[0, 0] = -1
x
```

[19]:
```
array([[-1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

[20]:
```
# Il est possible de remplacer plusieurs éléments par la même valeur d'un seul coup.
x[:, 0] = 0
x
```

[20]:
```
array([[ 0,  2,  3,  4],
       [ 0,  6,  7,  8],
       [ 0, 10, 11, 12]])
```

[21]:
```
# Toutes les fonctionnalités des slices sont disponibles: arr[start:stop:step]
x[:, ::2]
```

[21]:
```
array([[ 0,  3],
       [ 0,  7],
       [ 0, 11]])
```

[22]:
```
# Un accès *très* utile : l'indexation par tableau de booléens
a = np.random.random((5,4))
a
```

[22]:
```
array([[0.10193638, 0.31366494, 0.58893349, 0.62615365],
       [0.69863562, 0.22738459, 0.85689817, 0.20049676],
       [0.73153055, 0.7271929 , 0.74053103, 0.70424826],
       [0.07807063, 0.90004515, 0.83373539, 0.57301106],
       [0.99646386, 0.19844358, 0.83802383, 0.63492711]])
```

[23]:
```
# Admettons : on veut tronquer les valeurs inférieures à 0.5.
# On commence par se créer un "masque"
small = a < 0.5
small
```

[23]:
```
array([[ True,  True, False, False],
       [False,  True, False,  True],
       [False, False, False, False],
       [ True, False, False, False],
       [False,  True, False, False]])
```

[24]:
```
# On accède au array par le "masque"...
a[small] = 0
# ... et le tour est joué!
a
```

[24]:
```
array([[0.        , 0.        , 0.58893349, 0.62615365],
       [0.69863562, 0.        , 0.85689817, 0.        ],
       [0.73153055, 0.7271929 , 0.74053103, 0.70424826],
       [0.        , 0.90004515, 0.83373539, 0.57301106],
       [0.99646386, 0.        , 0.83802383, 0.63492711]])
```

5.2.3 Arithmétique

Les opérations arithmétiques sur `array` suivent la convention de l'algèbre linéaire (et sont donc plus intuitive).

[25]:
```
# Créons un array
x = np.arange(5)
x
```

[25]:
```
array([0, 1, 2, 3, 4])
```

[26]:
```
# Les opérations entre un array et un nombre sont effectuées sur tous les éléments du
↪array
# Exemple de la multiplication :
# (pour rappel l'opération float * list dans Python duplique la liste)
2.5 * x
```

[26]:
```
array([ 0. ,  2.5,  5. ,  7.5, 10. ])
```

[27]:
```
# Les opérations entre array de même taille s'effectuent élément par élément.
y = np.array([10, 11, 12, 13, 14])
```

[28]:
```
# Les opérations entre array de même taille s'effectuent élément par élément.
x + y
```

[28]:
```
array([10, 12, 14, 16, 18])
```

[29]:
```
# Les opérations entre array de même taille s'effectuent élément par élément.
x * y
```

[29]:
```
array([ 0, 11, 24, 39, 56])
```

[30]:
```
# Numpy dispose de nombreuses fonctions mathématiques : trigo, log, exp, ...
# Les fonctions de Numpy peuvent être appelées sur des array, auquel cas l'opération est appliquée sur tous les éléments.
np.sqrt(y)
```

[30]:
```
array([3.16227766, 3.31662479, 3.46410162, 3.60555128, 3.74165739])
```

Toutes les fonctions disponibles : http://docs.scipy.org/doc/numpy/reference/ufuncs.html#available-ufuncs

[31]:
```
# Une dernière remarque : numpy peut traiter les opérations arithmétiques entre array de dimensions différentes,
# on parle de "broadcasting".
# Exemple d'application au produit tensoriel :
# On redimensionne x pour avoir un vecteur ligne.
x = x.reshape((1,5))
# On redimensionne y pour avoir un vecteur colonne.
y = y.reshape((5,1))
# Leur produit donne un array de dimensions (5,5).
x*y
```

[31]:
```
array([[ 0, 10, 20, 30, 40],
       [ 0, 11, 22, 33, 44],
       [ 0, 12, 24, 36, 48],
       [ 0, 13, 26, 39, 52],
       [ 0, 14, 28, 42, 56]])
```

[32]:
```
from IPython.display import Image
Image(width=600, url='https://scipy-lectures.github.io/_images/numpy_broadcasting.png')
# source: http://scipy-lectures.github.io
```

[32]:
```
<IPython.core.display.Image object>
```

5.2.4 Changer la forme

Il est possible de changer la forme (*shape*) d'un array sans faire de copie (mais pas toujours)

[33]:
```
x = np.arange(6)
x
```

[33]:
```
array([0, 1, 2, 3, 4, 5])
```

[34]:
```
# on peut voir le contenu de x sous la forme d'un array 2d
y = x.reshape((2, 3))
y
```

[34]:
```
array([[0, 1, 2],
       [3, 4, 5]])
```

[35]:
```
# l'information est partagée, pas copiée, on parle de différentes 'views' sur la même donnée.
# modifier le contenu de x a un effet sur y
x[0] = -1
print(x)
print(y)
```

```
[-1  1  2  3  4  5]
[[-1  1  2]
 [ 3  4  5]]
```

[36]:
```
# on peut aussi utiliser l'indexation pour ajouter des dimensions
x[:, np.newaxis]
```

[36]:
```
array([[-1],
       [ 1],
       [ 2],
       [ 3],
       [ 4],
       [ 5]])
```

[37]:
```
# ce comportement se combine bien avec le broadcasting
a = np.arange(3)
b = np.arange(5)
a[:, np.newaxis] + b[np.newaxis, :]
```

[37]:
```
array([[0, 1, 2, 3, 4],
       [1, 2, 3, 4, 5],
       [2, 3, 4, 5, 6]])
```

[38]:
```
# on peut modifier directement la forme d'un array
x.shape = (3, 2)
x
```

[38]:
```
array([[-1,  1],
       [ 2,  3],
       [ 4,  5]])
```

5.2.5 Opérations sur les arrays

Quelques fonctions utiles parmi d'autres : np.where(), np.sum(), np.maximum(), np.minimum()

np.where() : « mélanger » deux arrays suivant une condition

[39]:
```
x = np.arange(10)
x
```

[39]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

[40]:
```
np.where(x<5, 0, 1)
```

[40]:
```
array([0, 0, 0, 0, 0, 1, 1, 1, 1, 1])
```

np.sum() : sommer un array selon un axe

[41]:
```
x = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
x
```

[41]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

[42]:
```
np.sum(x, axis=0)
```
[42]:
```
array([15, 18, 21, 24])
```

[43]:
```
np.sum(x, axis=1)
```
[43]:
```
array([10, 26, 42])
```

np.maximum(a, b) : construit un array composé du maximum entre a et b (avec du broadcasting)

[44]:
```
x = np.arange(10)
x
```
[44]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

[45]:
```
np.maximum(x, 4)
```
[45]:
```
array([4, 4, 4, 4, 4, 5, 6, 7, 8, 9])
```

[46]:
```
np.minimum(x, 4)
```
[46]:
```
array([0, 1, 2, 3, 4, 4, 4, 4, 4, 4])
```

5.2.6 Exercices

5.3 Scipy

Scipy est une collection d'algorithmes mathématiques et de fonctions utiles s'appuyant sur Numpy.

Voir un aperçu des fonctionnalités : http://docs.scipy.org/doc/scipy/reference/

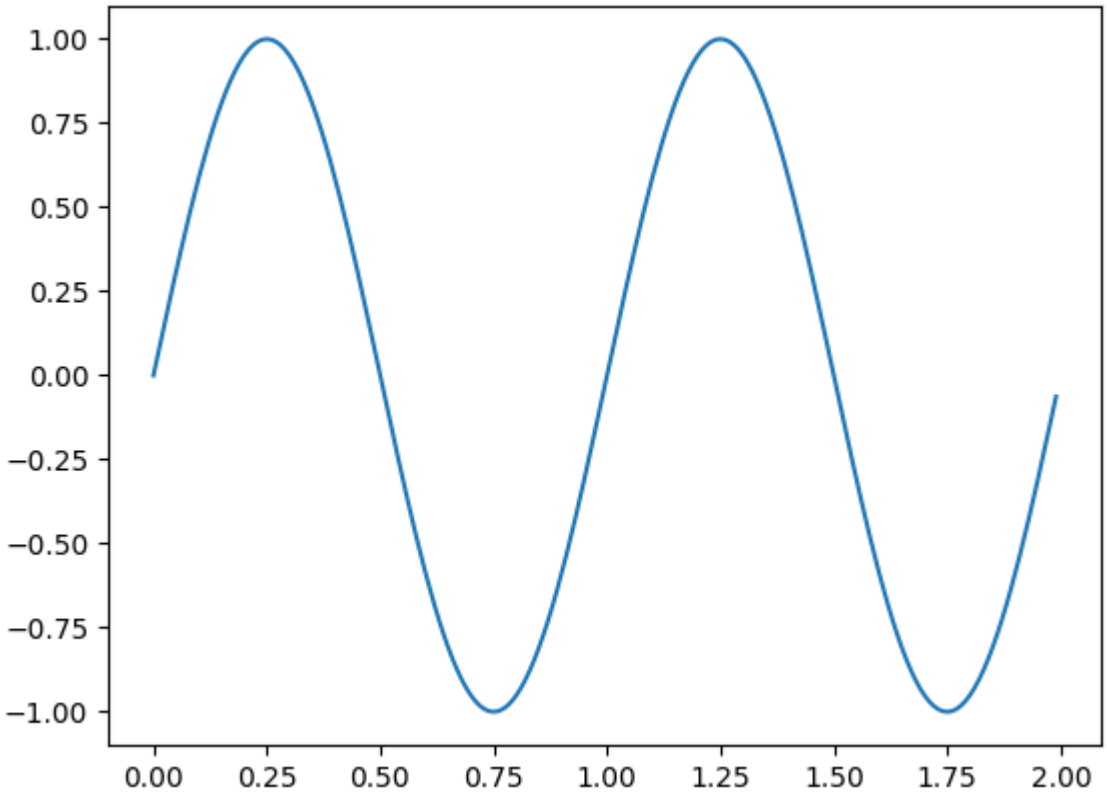
5.4 Matplotlib

Vous trouverez un très bon tutoriel pour un premier contact avec matplotlib ici : http://www.loria.fr/~rougier/teaching/matplotlib/

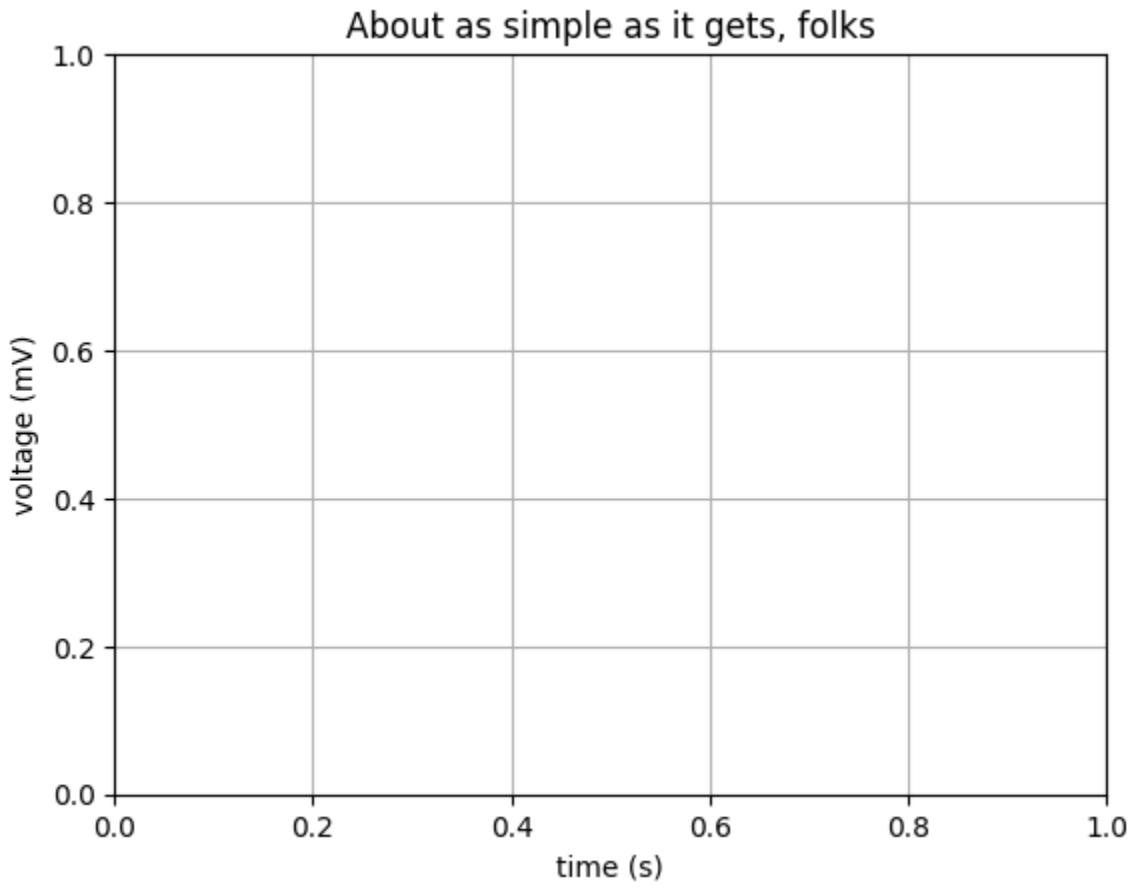
[1]:
```
import numpy as np  # numpy pour créer et manipuler des données
import matplotlib.pyplot as plt  # la formule magique
```

[2]:
```
# on se fabrique quelques données
t = np.arange(0.0, 2.0, 0.01)
s = np.sin(2 * np.pi * t)
```

[3]:
```
# on trace une courbe sommaire

plt.plot(t, s)

plt.show()  # cette ligne provoque l'affichage de la courbe
```



[4]:
```
# on peut évidemment décorer un peu tout ça
plt.xlabel('time (s)')
plt.ylabel('voltage (mV)')
plt.title('About as simple as it gets, folks')
plt.grid(True)

plt.show()
```



[5]:
```
# enregistrement
plt.savefig("test.png")
```

```
<Figure size 640x480 with 0 Axes>
```

Pour aller plus loin : http://www.loria.fr/~rougier/teaching/matplotlib/

Pour trouver l'inspiration : http://matplotlib.org/gallery.html

prev : Matplotlib | *home*

5.5 Pandas

https://pandas.pydata.org/

Pandas permet de travailler facilement avec certains types de données :

- — données tabulaires. ex : table SQL ou feuille Excel.
- — séries temporelles
- — et d'autres …

Les données peuvent être chargées depuis différentes sources (csv, excel, sql, …) et sont exposées principalement par deux structures de données : `Series` (1D) et `DataFrame` (2D).

Pandas apporte en particulier une réponse pour :

— la gestion des dates
— la gestion des données manquantes

note : les données utilisées sont issues de https://public.opendatasoft.com/explore/dataset/donnees-synop-essentielles-omm

[1]:
```
import pandas as pd
import matplotlib
```

[2]:
```
df = pd.read_csv("donnees-synop-essentielles-omm.csv", sep=';')
df.Date = pd.to_datetime(df.Date, utc=True)
df.head(5)
```

[2]:

	ID OMM station	Date	Pression au niveau mer	Variation de pression en 3 heures	Type de tendance barométrique	Direction du vent moyen 10 mn	Vitesse du vent moyen 10 mn	Température	Point de rosée	Humidité	…	Type nuage 3	Hauteur de base 3	Nébulosité couche nuageuse 4	Type nuage 4	Hauteur de base 4	Coordonnees	Nom	Type de tendance barométrique.1	Temps passé 1.1
0	7240	2018-08-08 15:00:00+00:00	101380	-130.0	7.0	250.0	4.0	298.65	285.45	44	…	NaN	NaN	NaN	NaN	NaN	47.4445, 0.727333	TOURS	7.0	NaN
1	7240	2018-08-09 18:00:00+00:00	102000	200.0	2.0	250.0	2.9	289.45	286.75	84	…	NaN	2940.0	NaN	NaN	NaN	47.4445, 0.727333	TOURS	2.0	NaN
2	7240	2018-08-10 06:00:00+00:00	102500	120.0	3.0	250.0	2.3	285.85	284.25	90	…	NaN	NaN	NaN	NaN	NaN	47.4445, 0.727333	TOURS	3.0	NaN
3	7240	2018-08-15 15:00:00+00:00	102030	-150.0	8.0	320.0	2.1	298.05	285.95	47	…	NaN	NaN	NaN	NaN	NaN	47.4445, 0.727333	TOURS	8.0	NaN
4	7240	2018-08-18 03:00:00+00:00	102440	10.0	0.0	330.0	0.7	285.65	283.15	85	…	NaN	NaN	NaN	NaN	NaN	47.4445, 0.727333	TOURS	0.0	NaN

(suite sur la page suivante)

(suite de la page précédente)

```
   Temps présent.1
0                0
1                0
2                0
3                0
4                0

[5 rows x 64 columns]
```

[3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26071 entries, 0 to 26070
Data columns (total 64 columns):
 #   Column                                                Non-Null Count  Dtype
---  ------                                                --------------  -----
 0   ID OMM station                                        26071 non-null  int64
 1   Date                                                  26071 non-null ␣
↪datetime64[ns, UTC]
 2   Pression au niveau mer                                26071 non-null  int64
 3   Variation de pression en 3 heures                     26062 non-null ␣
↪float64
 4   Type de tendance barométrique                         26062 non-null ␣
↪float64
 5   Direction du vent moyen 10 mn                         26036 non-null ␣
↪float64
 6   Vitesse du vent moyen 10 mn                           26037 non-null ␣
↪float64
 7   Température                                           26071 non-null ␣
↪float64
 8   Point de rosée                                        26071 non-null ␣
↪float64
 9   Humidité                                              26071 non-null  int64
 10  Visibilité horizontale                                26069 non-null ␣
↪float64
 11  Temps présent                                         25999 non-null  object
 12  Temps passé 1                                         7958 non-null   object
 13  Temps passé 2                                         7114 non-null  ␣
↪float64
 14  Nebulosité totale                                     19664 non-null ␣
↪float64
 15  Nébulosité  des nuages de l' étage inférieur          18503 non-null ␣
↪float64
 16  Hauteur de la base des nuages de l'étage inférieur    18511 non-null ␣
↪float64
 17  Type des nuages de l'étage inférieur                  9864 non-null  ␣
↪float64
 18  Type des nuages de l'étage moyen                      8899 non-null  ␣
↪float64
 19  Type des nuages de l'étage supérieur                  8655 non-null  ␣
↪float64
 20  Pression station                                      26071 non-null  int64
```

(suite sur la page suivante)

(suite de la page précédente)

```
 21  Niveau barométrique                                  0 non-null      ␣
↪float64
 22  Géopotentiel                                        0 non-null      ␣
↪float64
 23  Variation de pression en 24 heures                   11702 non-null ␣
↪float64
 24  Température minimale sur 12 heures                   6514 non-null  ␣
↪float64
 25  Température minimale sur 24 heures                   0 non-null      ␣
↪float64
 26  Température maximale sur 12 heures                   6514 non-null  ␣
↪float64
 27  Température maximale sur 24 heures                   0 non-null      ␣
↪float64
 28  Température minimale du sol sur 12 heures            13485 non-null ␣
↪float64
 29  Méthode de mesure Température du thermomètre mouillé  0 non-null      ␣
↪float64
 30  Température du thermomètre mouillé                   0 non-null      ␣
↪float64
 31  Rafale sur les 10 dernières minutes                  11840 non-null ␣
↪float64
 32  Rafales sur une période                              26007 non-null ␣
↪float64
 33  Periode de mesure de la rafale                       26051 non-null ␣
↪float64
 34  Etat du sol                                          12572 non-null ␣
↪float64
 35  Hauteur totale de la couche de neige, glace, autre au sol  15887 non-null ␣
↪float64
 36  Hauteur de la neige fraîche                          10499 non-null ␣
↪float64
 37  Periode de mesure de la neige fraiche                9543 non-null  ␣
↪float64
 38  Précipitations dans la dernière heure                26007 non-null ␣
↪float64
 39  Précipitations dans les 3 dernières heures           26013 non-null ␣
↪float64
 40  Précipitations dans les 6 dernières heures           15246 non-null ␣
↪float64
 41  Précipitations dans les 12 dernières heures          15154 non-null ␣
↪float64
 42  Précipitations dans les 24 dernières heures          13254 non-null ␣
↪float64
 43  Phénomène spécial 1                                  14283 non-null ␣
↪float64
 44  Phénomène spécial 2                                  14273 non-null ␣
↪float64
 45  Phénomène spécial 3                                  8318 non-null  ␣
↪float64
 46  Phénomène spécial 4                                  61 non-null     ␣
↪float64
```

(suite sur la page suivante)

(suite de la page précédente)

```
 47  Nébulosité couche nuageuse 1                          15269 non-null ␣
↪float64
 48  Type nuage 1                                          9358 non-null  ␣
↪float64
 49  Hauteur de base 1                                     14635 non-null ␣
↪float64
 50  Nébulosité couche nuageuse 2                          6281 non-null  ␣
↪float64
 51  Type nuage 2                                          3490 non-null  ␣
↪float64
 52  Hauteur de base 2                                     6138 non-null  ␣
↪float64
 53  Nébulosité couche nuageuse 3                          1622 non-null  ␣
↪float64
 54  Type nuage 3                                          516 non-null   ␣
↪float64
 55  Hauteur de base 3                                     1622 non-null  ␣
↪float64
 56  Nébulosité couche nuageuse 4                          2 non-null     ␣
↪float64
 57  Type nuage 4                                          203 non-null   ␣
↪float64
 58  Hauteur de base 4                                     2 non-null     ␣
↪float64
 59  Coordonnees                                           26071 non-null  object
 60  Nom                                                   26071 non-null  object
 61  Type de tendance barométrique.1                       26062 non-null ␣
↪float64
 62  Temps passé 1.1                                       7958 non-null   object
 63  Temps présent.1                                       25999 non-null  object
dtypes: datetime64[ns, UTC](1), float64(53), int64(4), object(6)
memory usage: 12.7+ MB
```

[4]:
```
df.describe()
```

[4]:
```
       ID OMM station  Pression au niveau mer  \
count         26071.0            26071.000000
mean           7240.0           101727.889226
std               0.0              892.362431
min            7240.0            97010.000000
25%            7240.0           101230.000000
50%            7240.0           101770.000000
75%            7240.0           102300.000000
max            7240.0           104290.000000

       Variation de pression en 3 heures  Type de tendance barométrique  \
count                       26062.000000                   26062.000000
mean                            0.034533                       4.338654
std                           121.018307                       2.697870
min                          -940.000000                       0.000000
25%                           -70.000000                       2.000000
50%                             0.000000                       4.000000
```

(suite sur la page suivante)

(suite de la page précédente)

```
75%                            70.000000                      7.000000
max                          1200.000000                      8.000000

       Direction du vent moyen 10 mn  Vitesse du vent moyen 10 mn  \
count                    26036.000000                 26037.000000
mean                       183.604624                     3.326124
std                        105.004084                     1.848995
min                          0.000000                     0.000000
25%                         80.000000                     2.100000
50%                        200.000000                     3.100000
75%                        270.000000                     4.500000
max                        360.000000                    17.500000

        Température  Point de rosée      Humidité  Visibilité horizontale  \
count  26071.000000    26071.000000  26071.000000             26069.000000
mean     285.248964      280.782876     76.787657             26243.515670
std        7.135546        5.411169     17.055975             17644.988344
min      261.850000      258.150000     17.000000                30.000000
25%      280.150000      277.050000     66.000000             12000.000000
50%      284.950000      281.150000     81.000000             24000.000000
75%      290.050000      284.850000     91.000000             35870.000000
max      309.850000      294.450000    100.000000             75000.000000

       …  Nébulosité couche nuageuse 2  Type nuage 2  Hauteur de base 2  \
count  …                    6281.000000   3490.000000        6138.000000
mean   …                       6.078013      4.205731        2605.171065
std    …                       1.695055      2.609169        2263.468510
min    …                       1.000000      0.000000         180.000000
25%    …                       5.000000      3.000000         990.000000
50%    …                       7.000000      6.000000        1680.000000
75%    …                       7.000000      6.000000        3480.000000
max    …                       8.000000      9.000000        8000.000000

       Nébulosité couche nuageuse 3  Type nuage 3  Hauteur de base 3  \
count                   1622.000000    516.000000        1622.000000
mean                       6.835388      4.695736        2666.134402
std                        1.210954      3.110655        2028.994254
min                        1.000000      0.000000         360.000000
25%                        6.000000      3.000000        1230.000000
50%                        7.000000      6.000000        1800.000000
75%                        8.000000      8.000000        3300.000000
max                        8.000000      9.000000        8000.000000

       Nébulosité couche nuageuse 4  Type nuage 4  Hauteur de base 4  \
count                           2.0    203.000000           2.000000
mean                            6.0      8.216749        4350.000000
std                             0.0      0.821924        2333.452378
min                             6.0      0.000000        2700.000000
25%                             6.0      8.000000        3525.000000
50%                             6.0      8.000000        4350.000000
75%                             6.0      9.000000        5175.000000
max                             6.0      9.000000        6000.000000
```

(suite sur la page suivante)

(suite de la page précédente)

```
       Type de tendance barométrique.1
count                     26062.000000
mean                          4.338654
std                           2.697870
min                           0.000000
25%                           2.000000
50%                           4.000000
75%                           7.000000
max                           8.000000

[8 rows x 57 columns]
```

[5]:
```
df.shape
```

[5]:
```
(26071, 64)
```

[6]:
```
df[50:53]
```

[6]:
```
    ID OMM station                      Date  Pression au niveau mer  \
50            7240 2012-04-06 15:00:00+00:00                  100910
51            7240 2012-04-13 03:00:00+00:00                  100610
52            7240 2012-09-11 21:00:00+00:00                  101950

    Variation de pression en 3 heures  Type de tendance barométrique  \
50                             -200.0                             8.0
51                              -50.0                             6.0
52                              170.0                             1.0

    Direction du vent moyen 10 mn  Vitesse du vent moyen 10 mn  Température  \
50                            10.0                          4.6       287.25
51                           270.0                          1.5       276.75
52                           320.0                          2.6       287.65

    Point de rosée  Humidité  …  Type nuage 3 Hauteur de base 3  \
50          279.85        61  …           NaN               NaN
51          274.65        86  …           NaN               NaN
52          282.05        69  …           NaN               NaN

   Nébulosité couche nuageuse 4  Type nuage 4  Hauteur de base 4  \
50                           NaN           NaN                NaN
51                           NaN           NaN                NaN
52                           NaN           NaN                NaN

          Coordonnees    Nom  Type de tendance barométrique.1  \
50  47.4445, 0.727333  TOURS                              8.0
51  47.4445, 0.727333  TOURS                              6.0
52  47.4445, 0.727333  TOURS                              1.0

                                    Temps passé 1.1  Temps présent.1
50                                              NaN               10
51                                              NaN                1
```

(suite sur la page suivante)

(suite de la page précédente)

```
52  Nuages couvrant plus de la moitié du ciel pend…                2

[3 rows x 64 columns]
```

[7]:
```
df.columns
```

[7]:
```
Index(['ID OMM station', 'Date', 'Pression au niveau mer',
       'Variation de pression en 3 heures', 'Type de tendance barométrique',
       'Direction du vent moyen 10 mn', 'Vitesse du vent moyen 10 mn',
       'Température', 'Point de rosée', 'Humidité', 'Visibilité horizontale',
       'Temps présent', 'Temps passé 1', 'Temps passé 2', 'Nebulosité totale',
       'Nébulosité  des nuages de l' étage inférieur',
       'Hauteur de la base des nuages de l'étage inférieur',
       'Type des nuages de l'étage inférieur',
       'Type des nuages de l'étage moyen',
       'Type des nuages de l'étage supérieur', 'Pression station',
       'Niveau barométrique', 'Géopotentiel',
       'Variation de pression en 24 heures',
       'Température minimale sur 12 heures',
       'Température minimale sur 24 heures',
       'Température maximale sur 12 heures',
       'Température maximale sur 24 heures',
       'Température minimale du sol sur 12 heures',
       'Méthode de mesure Température du thermomètre mouillé',
       'Température du thermomètre mouillé',
       'Rafale sur les 10 dernières minutes', 'Rafales sur une période',
       'Periode de mesure de la rafale', 'Etat du sol',
       'Hauteur totale de la couche de neige, glace, autre au sol',
       'Hauteur de la neige fraîche', 'Periode de mesure de la neige fraiche',
       'Précipitations dans la dernière heure',
       'Précipitations dans les 3 dernières heures',
       'Précipitations dans les 6 dernières heures',
       'Précipitations dans les 12 dernières heures',
       'Précipitations dans les 24 dernières heures', 'Phénomène spécial 1',
       'Phénomène spécial 2', 'Phénomène spécial 3', 'Phénomène spécial 4',
       'Nébulosité couche nuageuse 1', 'Type nuage 1', 'Hauteur de base 1',
       'Nébulosité couche nuageuse 2', 'Type nuage 2', 'Hauteur de base 2',
       'Nébulosité couche nuageuse 3', 'Type nuage 3', 'Hauteur de base 3',
       'Nébulosité couche nuageuse 4', 'Type nuage 4', 'Hauteur de base 4',
       'Coordonnees', 'Nom', 'Type de tendance barométrique.1',
       'Temps passé 1.1', 'Temps présent.1'],
      dtype='object')
```

[8]:
```
df["Date"]
```

[8]:
```
0       2018-08-08 15:00:00+00:00
1       2018-08-09 18:00:00+00:00
2       2018-08-10 06:00:00+00:00
3       2018-08-15 15:00:00+00:00
4       2018-08-18 03:00:00+00:00
                   …
26066   2018-09-08 03:00:00+00:00
```

(suite sur la page suivante)

(suite de la page précédente)

```
26067   2018-12-12 03:00:00+00:00
26068   2018-12-05 15:00:00+00:00
26069   2018-12-04 00:00:00+00:00
26070   2018-11-29 12:00:00+00:00
Name: Date, Length: 26071, dtype: datetime64[ns, UTC]
```

[9]:
```
df["Date"].max()
```

[9]:
```
Timestamp('2019-01-30 21:00:00+0000', tz='UTC')
```

[10]:
```
df["Date"].min()
```

[10]:
```
Timestamp('2010-01-01 00:00:00+0000', tz='UTC')
```

[11]:
```
df[["Date", "Température"]].head()
```

[11]:
```
                       Date  Température
0 2018-08-08 15:00:00+00:00       298.65
1 2018-08-09 18:00:00+00:00       289.45
2 2018-08-10 06:00:00+00:00       285.85
3 2018-08-15 15:00:00+00:00       298.05
4 2018-08-18 03:00:00+00:00       285.65
```

[12]:
```
df["Température"].min()
```

[12]:
```
261.85
```

[13]:
```
df["Température"] == df["Température"].min()
```

[13]:
```
0        False
1        False
2        False
3        False
4        False
         …
26066    False
26067    False
26068    False
26069    False
26070    False
Name: Température, Length: 26071, dtype: bool
```

[14]:
```
df[df["Température"] == df["Température"].min()]
```

[14]:
```
       ID OMM station                      Date  Pression au niveau mer  \
21810            7240 2012-02-09 06:00:00+00:00                  103740

       Variation de pression en 3 heures  Type de tendance barométrique  \
21810                               10.0                            2.0

       Direction du vent moyen 10 mn  Vitesse du vent moyen 10 mn  \
21810                          310.0                          1.0
```

(suite sur la page suivante)

(suite de la page précédente)

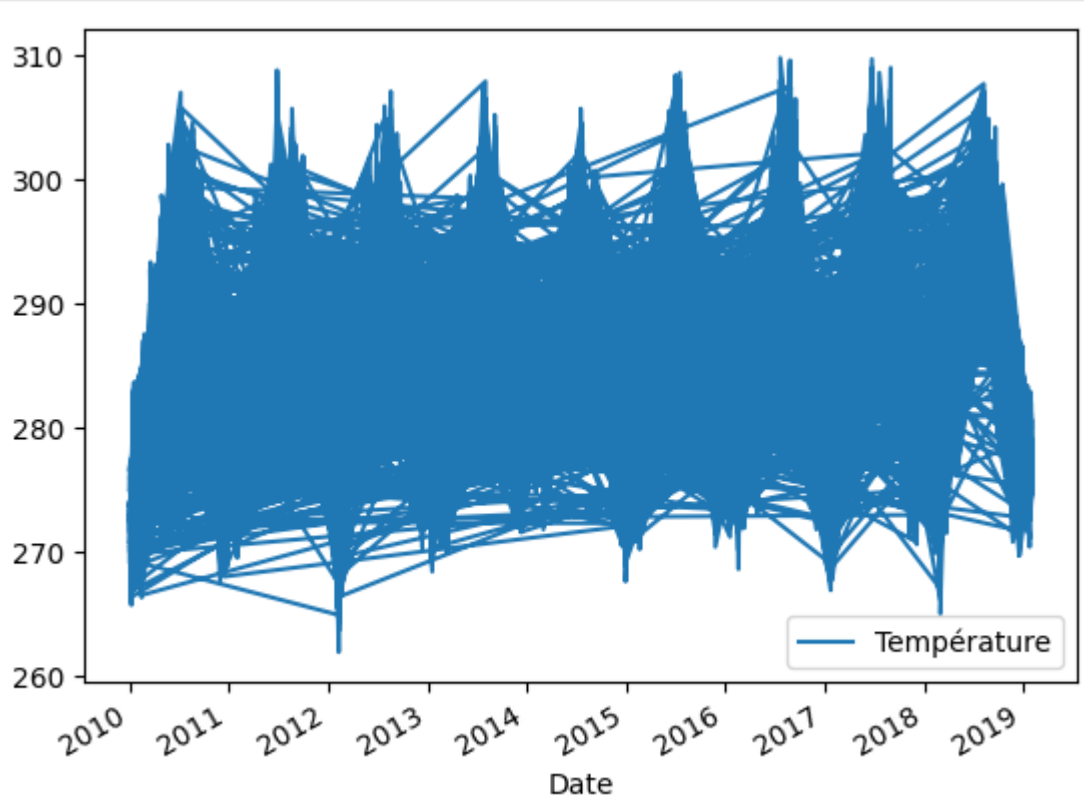
```
       Température  Point de rosée  Humidité  …  Type nuage 3  \
21810       261.85          260.55        89  …           NaN

      Hauteur de base 3 Nébulosité couche nuageuse 4  Type nuage 4  \
21810               NaN                           NaN           NaN

       Hauteur de base 4        Coordonnees    Nom  \
21810                NaN  47.4445, 0.727333  TOURS

       Type de tendance barométrique.1  Temps passé 1.1  Temps présent.1
21810                              2.0              NaN               10

[1 rows x 64 columns]
```

[15]:
```
df[df["Température"] == df["Température"].max()][["Date", "Température"]]
```

[15]:
```
                          Date  Température
7993 2016-07-19 15:00:00+00:00       309.85
```

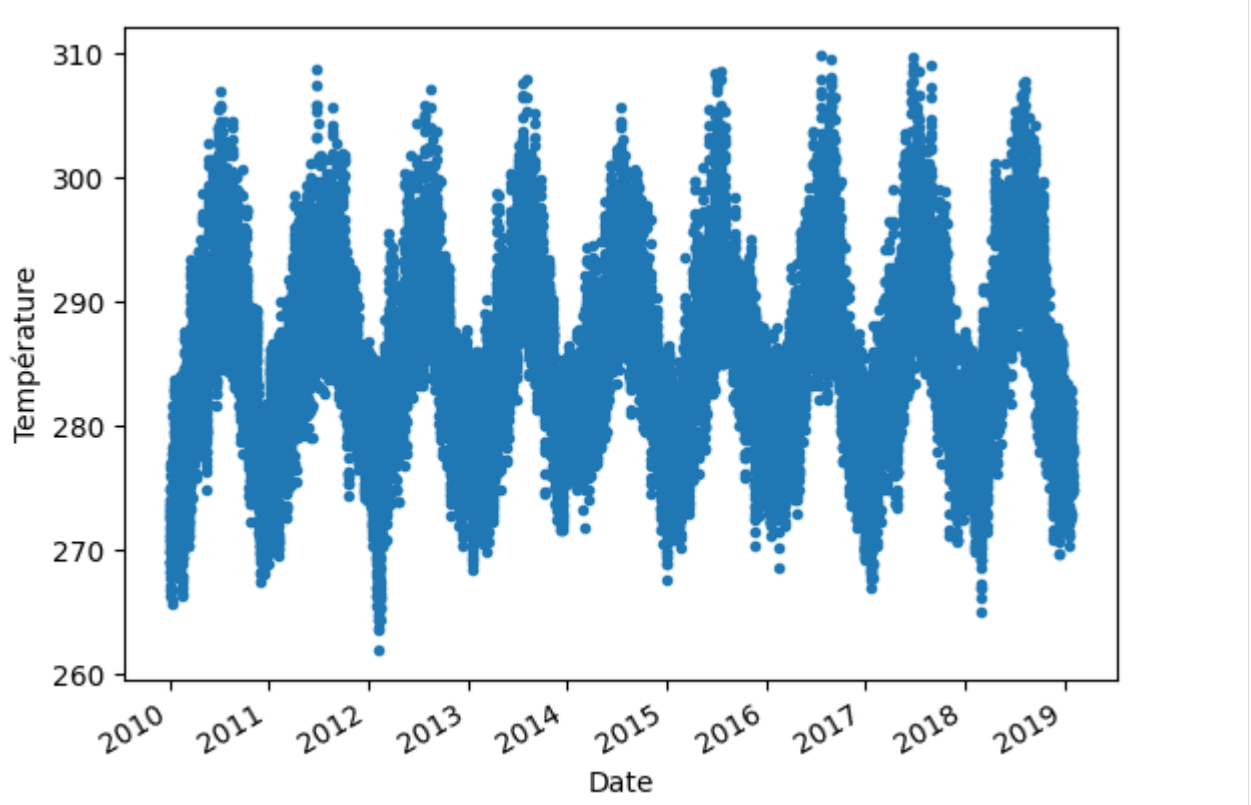
[16]:
```
df.plot(x="Date", y="Température")
```

[16]:
```
<Axes: xlabel='Date'>
```



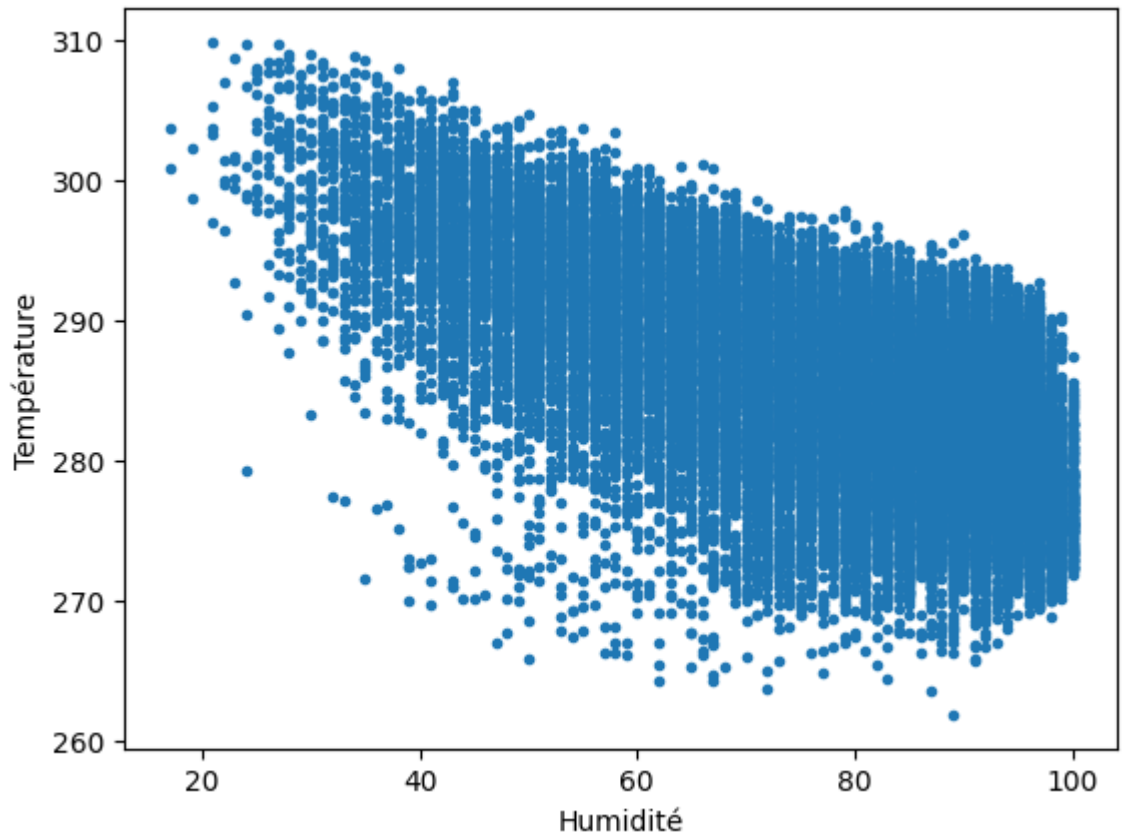
[17]:
```
ax = df.plot(x="Date", y="Température", linestyle="", marker=".", legend=False)
ax.set_ylabel('Température')
```

[17]:
```
Text(0, 0.5, 'Température')
```



[18]:
```
ax = df.plot(x="Humidité", y="Température", linestyle="", marker=".", legend=False)
ax.set_ylabel('Température')
```

[18]:
```
Text(0, 0.5, 'Température')
```



[19]:
```
tdf = df.set_index("Date").sort_index()
```

[20]:
```
tdf.head()
```

[20]:
```
                           ID OMM station  Pression au niveau mer  \
Date
2010-01-01 00:00:00+00:00            7240                   99320
2010-01-01 03:00:00+00:00            7240                   99400
2010-01-01 06:00:00+00:00            7240                   99610
2010-01-01 09:00:00+00:00            7240                   99950
2010-01-01 12:00:00+00:00            7240                  100250

                           Variation de pression en 3 heures  \
Date
2010-01-01 00:00:00+00:00                                0.0
2010-01-01 03:00:00+00:00                               70.0
2010-01-01 06:00:00+00:00                              210.0
2010-01-01 09:00:00+00:00                              330.0
2010-01-01 12:00:00+00:00                              300.0

                           Type de tendance barométrique  \
Date
2010-01-01 00:00:00+00:00                            5.0
2010-01-01 03:00:00+00:00                            3.0
```

(suite sur la page suivante)

(suite de la page précédente)

```
2010-01-01 06:00:00+00:00                            3.0
2010-01-01 09:00:00+00:00                            3.0
2010-01-01 12:00:00+00:00                            2.0

                           Direction du vent moyen 10 mn  \
Date
2010-01-01 00:00:00+00:00                           30.0
2010-01-01 03:00:00+00:00                           20.0
2010-01-01 06:00:00+00:00                          360.0
2010-01-01 09:00:00+00:00                          360.0
2010-01-01 12:00:00+00:00                          330.0

                           Vitesse du vent moyen 10 mn  Température  \
Date
2010-01-01 00:00:00+00:00                            4.6       276.55
2010-01-01 03:00:00+00:00                            4.6       274.95
2010-01-01 06:00:00+00:00                            5.1       274.05
2010-01-01 09:00:00+00:00                            4.6       273.65
2010-01-01 12:00:00+00:00                            5.7       273.65

                           Point de rosée  Humidité  Visibilité horizontale  \
Date
2010-01-01 00:00:00+00:00          275.55        93                  4000.0
2010-01-01 03:00:00+00:00          273.95        93                  2600.0
2010-01-01 06:00:00+00:00          272.55        90                  7000.0
2010-01-01 09:00:00+00:00          271.75        87                  6000.0
2010-01-01 12:00:00+00:00          270.95        82                 10000.0

                           … Type nuage 3 Hauteur de base 3  \
Date                       …
2010-01-01 00:00:00+00:00  …           NaN               NaN
2010-01-01 03:00:00+00:00  …           NaN               NaN
2010-01-01 06:00:00+00:00  …           NaN               NaN
2010-01-01 09:00:00+00:00  …           NaN               NaN
2010-01-01 12:00:00+00:00  …           NaN               NaN

                           Nébulosité couche nuageuse 4  Type nuage 4  \
Date
2010-01-01 00:00:00+00:00                           NaN           NaN
2010-01-01 03:00:00+00:00                           NaN           NaN
2010-01-01 06:00:00+00:00                           NaN           NaN
2010-01-01 09:00:00+00:00                           NaN           NaN
2010-01-01 12:00:00+00:00                           NaN           NaN

                           Hauteur de base 4        Coordonnees    Nom  \
Date
2010-01-01 00:00:00+00:00               NaN  47.4445, 0.727333  TOURS
2010-01-01 03:00:00+00:00               NaN  47.4445, 0.727333  TOURS
2010-01-01 06:00:00+00:00               NaN  47.4445, 0.727333  TOURS
2010-01-01 09:00:00+00:00               NaN  47.4445, 0.727333  TOURS
2010-01-01 12:00:00+00:00               NaN  47.4445, 0.727333  TOURS
```

(suite sur la page suivante)

(suite de la page précédente)

```
                           Type de tendance barométrique.1  Temps passé 1.1  \
Date
2010-01-01 00:00:00+00:00                              5.0            Pluie
2010-01-01 03:00:00+00:00                              3.0              NaN
2010-01-01 06:00:00+00:00                              3.0              NaN
2010-01-01 09:00:00+00:00                              3.0              NaN
2010-01-01 12:00:00+00:00                              2.0              NaN

                           Temps présent.1
Date
2010-01-01 00:00:00+00:00               60
2010-01-01 03:00:00+00:00               10
2010-01-01 06:00:00+00:00               20
2010-01-01 09:00:00+00:00                0
2010-01-01 12:00:00+00:00                0

[5 rows x 63 columns]
```

[21]:
```
df.head()
```

[21]:
```
   ID OMM station                      Date  Pression au niveau mer  \
0            7240 2018-08-08 15:00:00+00:00                  101380
1            7240 2018-08-09 18:00:00+00:00                  102000
2            7240 2018-08-10 06:00:00+00:00                  102500
3            7240 2018-08-15 15:00:00+00:00                  102030
4            7240 2018-08-18 03:00:00+00:00                  102440

   Variation de pression en 3 heures  Type de tendance barométrique  \
0                             -130.0                            7.0
1                              200.0                            2.0
2                              120.0                            3.0
3                             -150.0                            8.0
4                               10.0                            0.0

   Direction du vent moyen 10 mn  Vitesse du vent moyen 10 mn  Température  \
0                          250.0                          4.0       298.65
1                          250.0                          2.9       289.45
2                          250.0                          2.3       285.85
3                          320.0                          2.1       298.05
4                          330.0                          0.7       285.65

   Point de rosée  Humidité  …  Type nuage 3 Hauteur de base 3  \
0          285.45        44  …           NaN               NaN
1          286.75        84  …           NaN            2940.0
2          284.25        90  …           NaN               NaN
3          285.95        47  …           NaN               NaN
4          283.15        85  …           NaN               NaN

  Nébulosité couche nuageuse 4  Type nuage 4  Hauteur de base 4  \
0                          NaN           NaN                NaN
1                          NaN           NaN                NaN
2                          NaN           NaN                NaN
```

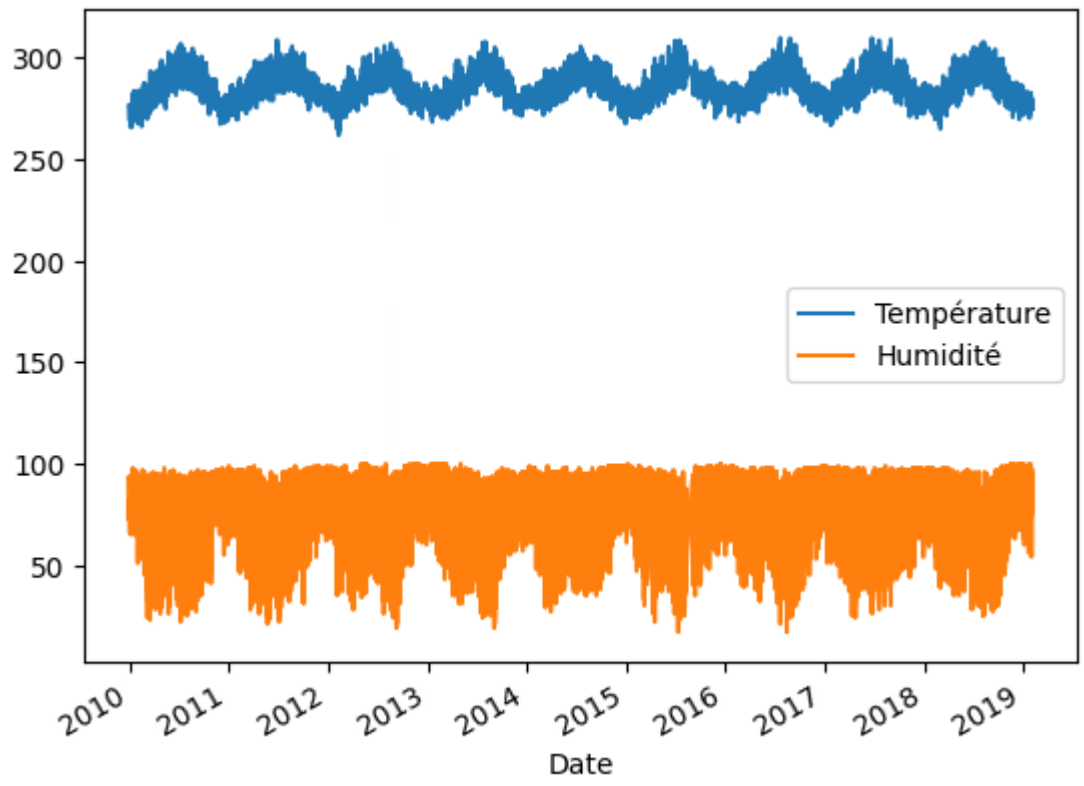
(suite sur la page suivante)

(suite de la page précédente)

```
3                          NaN           NaN                NaN
4                          NaN           NaN                NaN

         Coordonnees    Nom  Type de tendance barométrique.1  Temps passé 1.1  \
0  47.4445, 0.727333  TOURS                              7.0              NaN
1  47.4445, 0.727333  TOURS                              2.0              NaN
2  47.4445, 0.727333  TOURS                              3.0              NaN
3  47.4445, 0.727333  TOURS                              8.0              NaN
4  47.4445, 0.727333  TOURS                              0.0              NaN

   Temps présent.1
0                0
1                0
2                0
3                0
4                0

[5 rows x 64 columns]
```

[22]:
```
tdf.plot(y=["Température", "Humidité"])
```

[22]:
```
<Axes: xlabel='Date'>
```



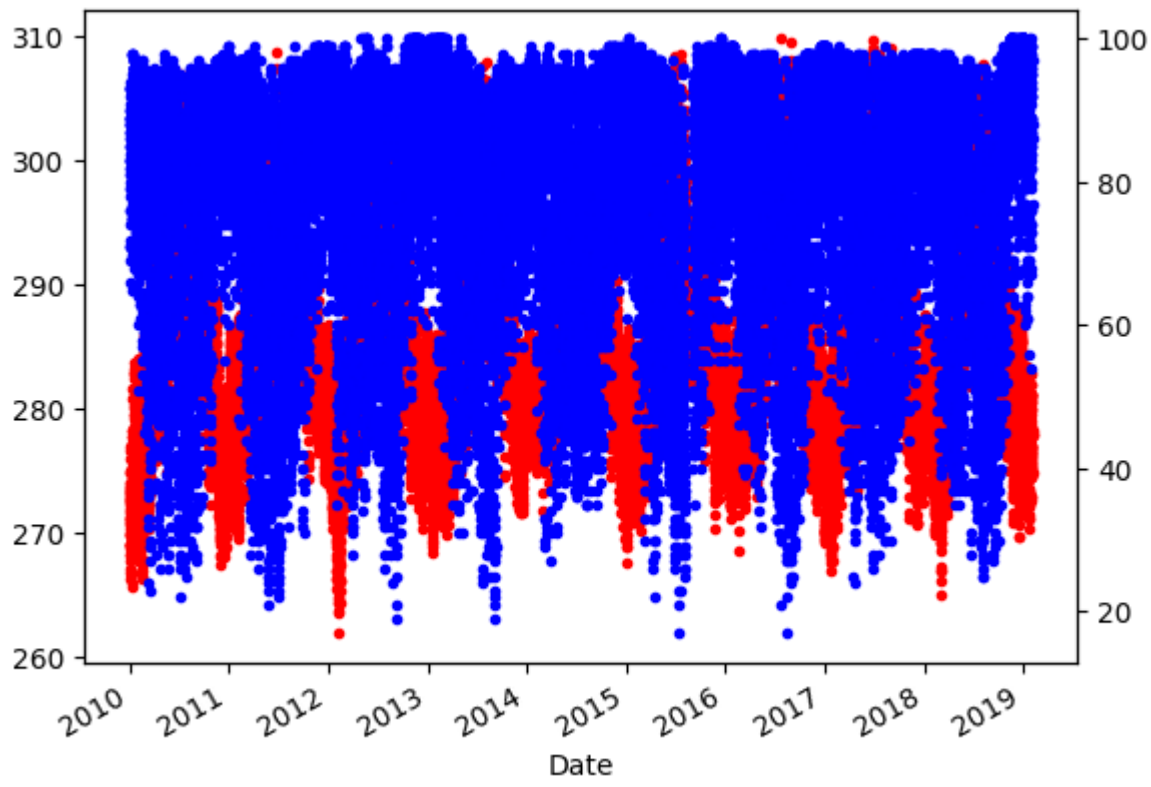
[23]:
```
import matplotlib.pyplot as plt

fig, ax1 = plt.subplots()
```

(suite sur la page suivante)

(suite de la page précédente)

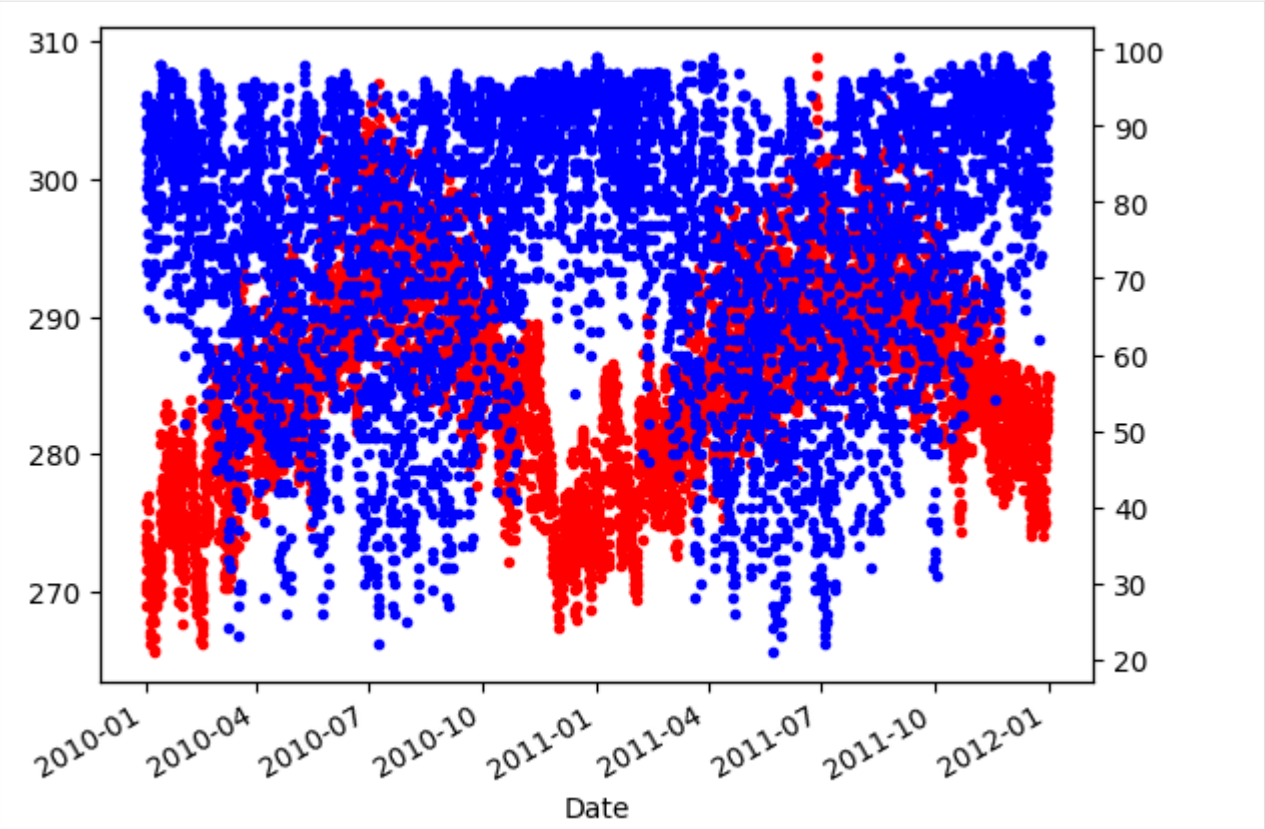
```
ax2 = ax1.twinx()

tdf.plot(y="Température", ax=ax1, legend=False, linestyle="", marker=".", color="r")
tdf.plot(y="Humidité", ax=ax2, legend=False, linestyle="", marker=".", color="b")
```

[23]:
```
<Axes: xlabel='Date'>
```



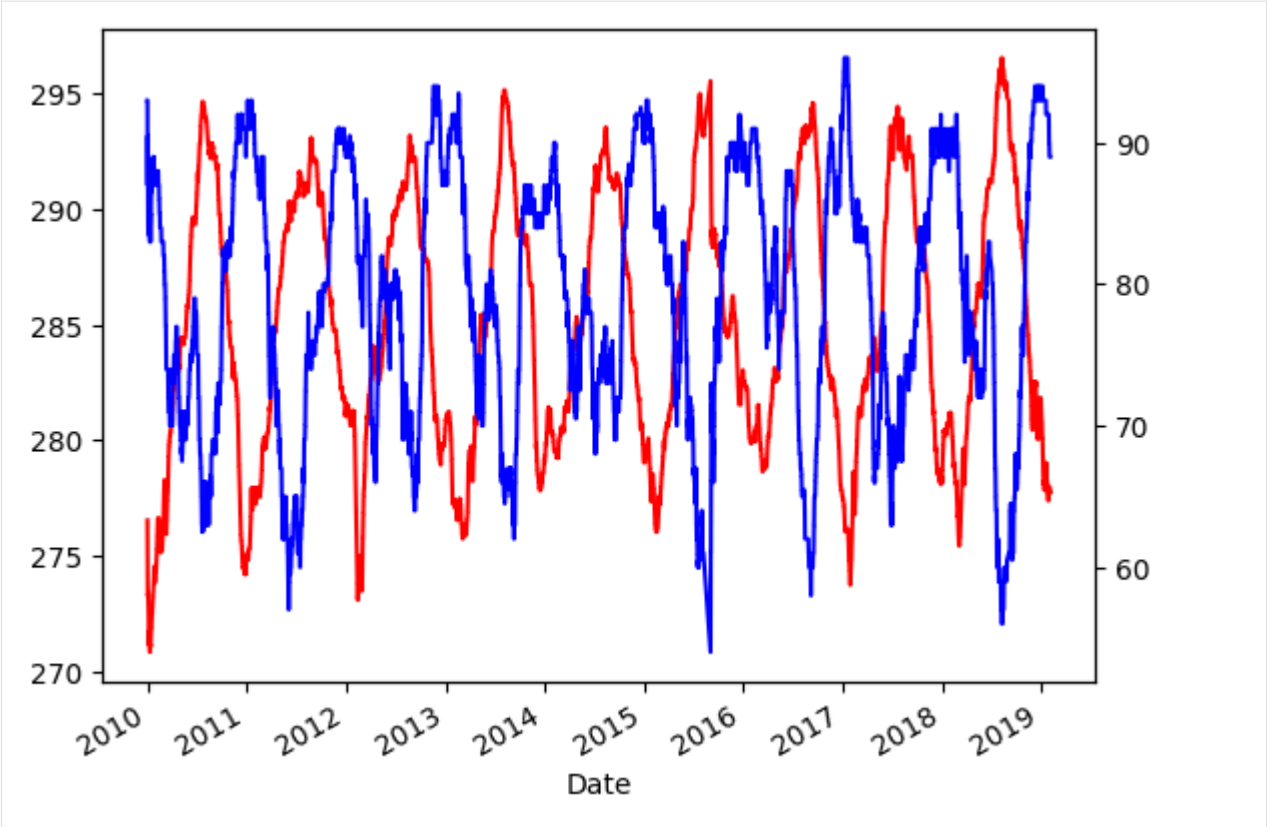
[24]:
```
import matplotlib.pyplot as plt

fig, ax1 = plt.subplots()
ax2 = ax1.twinx()

tdf['2010':'2011'].plot(y="Température", ax=ax1, legend=False, linestyle="", marker="."
↪", color="r")
tdf['2010':'2011'].plot(y="Humidité", ax=ax2, legend=False, linestyle="", marker=".",␣
↪color="b")
```

[24]:
```
<Axes: xlabel='Date'>
```



[25]:
```
import matplotlib.pyplot as plt

fig, ax1 = plt.subplots()
ax2 = ax1.twinx()

rtdf = tdf[["Température", "Humidité"]].rolling("30d").median()

rtdf.plot(y="Température", ax=ax1, legend=False, color="r")
rtdf.plot(y="Humidité", ax=ax2, legend=False, color="b")
```

[25]:
```
<Axes: xlabel='Date'>
```


295
290
285
280
275
270
90
80
70
60
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
Date

6 Exercices

6.1 Banque d'exercices

Cette page regroupe les exercices proposés au gré de la formation.

Certains des exercices sont des classiques revisités et d'autres sont librement inspirés d"ici (https://inventwithpython.com/pythongently/) et de là (https://dabeaz-course.github.io/practical-python/Notes/Contents.html)

6.1.1 Exercice 1 : double

Énoncé

Section *Introduction aux fonctions*

Écrire la fonction `double(x)` qui renvoie le double de `x`.

Vérifier les résultats de cette fonction avec `assert`, par exemple :

```
assert double(21) == 42
assert double(0) == 0
assert double(-99) == -198
assert double(3150) == 6300
```

Voir ce qui se passe avec

```
assert double(5)==9
```

Solution

[5]:
```
def double(x):
    return 2*x
```

[6]:
```
assert double(21) == 42
assert double(0) == 0
assert double(-99) == -198
assert double(3150) == 6300
```

[7]:
```
assert double(5)==9
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Cell In[7], line 1
----> 1 assert double(5)==9

AssertionError:
```

6.1.2 Exercice 2 : conversion de température

Énoncé

Section *Introduction aux fonctions*

Écrire la fonction `to_celsius(temp)` qui renvoie en °C la température `temp` donnée en °F.

*La formule est donnée par $ Celsius = :nbsphinx-math :*frac{5}{9}`(Fahrenheit - 32) $ <https://fr.wikipedia.org/wiki/Degr%C3%A9_Fahrenheit>`__.

Écrire la fonction inverse `to_farenheit(temp)`.

Vérifier les résultats :

```
assert to_celsius(32) == 0
assert to_farenheit(100) == 212
assert to_celsius(-40) == -40
assert to_farenheit(-40) == -40
assert to_celsius(to_farenheit(15.)) == 15.
```

source de l'exercice (http://inventwithpython.com/pythongently/exercise2/)

Solution

[15]:
```
def to_celsius(temp):
    return 5/9 * (temp-32)

def to_farenheit(temp):
    return temp * 9/5 +32
```

[16]:
```
assert to_celsius(32) == 0
assert to_farenheit(100) == 212
assert to_celsius(-40) == -40
assert to_farenheit(-40) == -40
assert to_celsius(to_farenheit(15.)) == 15.
```

6.1.3 Exercice 3 : conversion de secondes

Énoncé

Section *Introduction aux fonctions*

Écrire une fonction `h_m_s(seconds)` qui convertit un nombre total de secondes `h_m_s(seconds)` en heures, minutes et secondes sous la forme d'une chaine de caractères. Les nombres seront des entiers, et les secondes restantes seront arrondies à l'entier inférieur.

Vérifier les résultats :

```
assert h_m_s(0) == '0h 0m 0s'
assert h_m_s(60) == '0h 1m 0s'
assert h_m_s(3600) == '1h 0m 0s'
assert h_m_s(5700) == '1h 35m 0s'
assert h_m_s(18453) == '5h 7m 33s'
assert h_m_s(60.12586) == '0h 1m 0s'
```

Étape 2

Section *Structure de contrôle*

Mettre à jour la fonction pour qu'elle n'affiche les heures et minutes que si nécessaire.

```
assert h_m_s(0) == '00s'
assert h_m_s(60) == '01m 00s'
assert h_m_s(3600) == '1h 00m 00s'
assert h_m_s(5700) == '1h 35m 00s'
assert h_m_s(18453) == '5h 07m 33s'
assert h_m_s(60.12586) == '01m 00s'
```

Étape 3

Section *Utilisation avancée des chaines de caractères*

Mettre à jour la fonction afin qu'elle affiche les minutes et les secondes sur 2 caractères.

```
assert h_m_s(0) == '00s'
assert h_m_s(60) == '01m 00s'
assert h_m_s(3600) == '1h 00m 00s'
assert h_m_s(5700) == '01h 35m 00s'
assert h_m_s(18453) == '05h 07m 33s'
assert h_m_s(60.12586) == '01m 00s'
```

source de l'exercice (http://inventwithpython.com/pythongently/exercise11/)

Solution (étape 1)

[34]:
```
def h_m_s(seconds):
    seconds = int(seconds)
    hours = seconds//3600
    seconds = seconds%3600
    minutes = seconds//60
    seconds = seconds%60
    result = str(hours) + 'h ' + str(minutes) + 'm ' + str(seconds) + 's'
    return result
```

[35]:
```
assert h_m_s(0) == '0h 0m 0s'
assert h_m_s(60) == '0h 1m 0s'
assert h_m_s(3600) == '1h 0m 0s'
assert h_m_s(5700) == '1h 35m 0s'
assert h_m_s(18453) == '5h 7m 33s'
assert h_m_s(60.12586) == '0h 1m 0s'
```

Solution (étape 2, avec `if`)

[36]:
```
def h_m_s(seconds):
    seconds = int(seconds)
    hours = seconds//3600
    seconds = seconds%3600
    minutes = seconds//60
    seconds = seconds%60
    if hours > 0:
        result = str(hours) + 'h ' + str(minutes) + 'm '
    elif minutes > 0:
        result = str(minutes) + 'm '
    else:
        result = ''
    result = result + str(seconds) + 's'
    return result
```

[37]:
```
assert h_m_s(0) == '0s'
assert h_m_s(60) == '1m 0s'
assert h_m_s(3600) == '1h 0m 0s'
assert h_m_s(5700) == '1h 35m 0s'
assert h_m_s(18453) == '5h 7m 33s'
assert h_m_s(60.12586) == '1m 0s'
```

Solution (étape 3, avec `f-string`)

[38]:
```
def h_m_s(seconds):
    seconds = int(seconds)
    hours = seconds//3600
    seconds = seconds%3600
    minutes = seconds//60
    seconds = seconds%60
    if hours > 0:
        hours = f'{hours}h '
    else:
        hours = ''
    if minutes > 0 or hours:
        minutes = f'{minutes:>02}m '
    else:
        minutes = ''
    seconds = f'{seconds:>02}s'
    return f'{hours}{minutes}{seconds}'
```

[39]:
```
assert h_m_s(0) == '00s'
assert h_m_s(60) == '01m 00s'
assert h_m_s(3600) == '1h 00m 00s'
assert h_m_s(5700) == '1h 35m 00s'
assert h_m_s(18453) == '5h 07m 33s'
assert h_m_s(60.12586) == '01m 00s'
```

6.1.4 Exercice 4 : liste de zéros

Énoncé

Section *Les conteneurs de base*

Créer une fonction `make_zeros_list(nzeros)` qui renvoie une liste avec `nzeros` 0 sous la forme de flottants.

La fonction s'assurera que `nzeros` est un entier positif mais traitera le cas `nzeros=0`.

Vérifier les résultats :

```
assert make_zeros_list(5) == [0., 0., 0., 0., 0.]
assert make_zeros_list(1) == [0.]
assert make_zeros_list(0) == []
```

Solution

[1]:
```
def make_zeros_list(nzeros):
    assert isinstance(nzeros, int)
    assert nzeros >= 0
    return [0.]*nzeros
```

[2]:
```
assert make_zeros_list(5) == [0., 0., 0., 0., 0.]
assert make_zeros_list(1) == [0.]
assert make_zeros_list(0) == []
```

6.1.5 Exercice 5 : compte occurences

Énoncé

Section *Structure de contrôle*

Ecrire une fonction `compte_occurences(chaîne)` qui compte le nombre d'occurrences de chaque caractère dans la chaîne de caractères `chaîne`. La fonction retournera un `dict` où les clés seront les caractères rencontrés et les valeurs, le nombre d'occurences de chaque caractère.

Vérifier les résultats :

```
assert compte_occurences('python') == {'h': 1,'n': 1, 'o': 1, 'p': 1, 't': 1, 'y': 1}
assert compte_occurences('cool') == {'c': 1, 'l': 1, 'o': 2}
```

Étape 2

Section *Utilisation avancée des conteneurs*

Mettre à jour la fonction en utilisant une/des méthode(s) des dictionnaires.

Vérifier les résultats (identiques aux précédents)

Bonus

Section *Qu'est-ce qu'un module ?*

La librairie standard ne proposerait-elle pas déjà quelque chose ? Au hasard, module collections (https://docs.python.org/3/library/collections.html) ?

Vérifier les résultats (identiques aux précédents)

Solution (étape 1)

[2]:
```
def compte_occurences(chaîne):
    mon_dico = {}
    for char in chaîne:
        if char in mon_dico:
            mon_dico[char] += 1
        else:
            mon_dico[char] = 1
    return mon_dico
```

[3]:
```
assert compte_occurences('python') == {'h': 1,'n': 1, 'o': 1, 'p': 1, 't': 1, 'y': 1}
assert compte_occurences('cool') == {'c': 1, 'l': 1, 'o': 2}
```

Solution (étape 2, avec méthode)

[25]:
```
def compte_occurences(chaîne):
    mon_dico = {}
    for char in chaîne:
        mon_dico[char] = mon_dico.get(char, 0) + 1
    return mon_dico

def compte_occurences(chaîne):
    # Variante
    mon_dico = {}
    for char in chaîne:
        mon_dico.setdefault(char, 0)
        mon_dico[char] += 1
    return mon_dico
```

[26]:
```
assert compte_occurences('python') == {'h': 1,'n': 1, 'o': 1, 'p': 1, 't': 1, 'y': 1}
assert compte_occurences('cool') == {'c': 1, 'l': 1, 'o': 2}
```

Solution (bonus)

[68]:
```
from collections import Counter as compte_occurences
```

[69]:
```
assert compte_occurences('python') == {'h': 1,'n': 1, 'o': 1, 'p': 1, 't': 1, 'y': 1}
assert compte_occurences('cool') == {'c': 1, 'l': 1, 'o': 2}
```

Pourtant, ce que retourne la fonction n'est pas un dictionnaire ?

[70]:
```
compte_occurences('python')
```

[70]:
```
Counter({'p': 1, 'y': 1, 't': 1, 'h': 1, 'o': 1, 'n': 1})
```

Mais, Python s'occupe de la conversion derrière le rideau lors du test de comparaison !

[71]:
```
dict(compte_occurences('python'))
```

[71]:
```
{'p': 1, 'y': 1, 't': 1, 'h': 1, 'o': 1, 'n': 1}
```

6.1.6 Exercice 6 : suite de Collatz

Énoncé

Section *Structure de contrôle*

Construire une fonction `collatz(n)` qui, partant d'un nombre entier `n`, construit la suite de Collatz (https://fr.wikipedia.org/wiki/Lothar_Collatz#Suite_de_Collatz) qui est définie selon :

```
         | u(n)/2   si u(n) est pair
u(n+1) = |
         | 3u(n)+1  sinon
```

La fonction vérifiera que `n` est un entier positif et retournera une liste avec les valeurs de la suite.

NB : arrêter la boucle de construction dès lors que `u(n) = 1` (conjecture de Collatz (https://fr.wikipedia.org/wiki/Conjecture_de_Syracuse))

Vérifier les résultats :

```
assert collatz(1) == [1]
assert collatz(10) == [10, 5, 16, 8, 4, 2, 1]
assert collatz(11) == [11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1]
assert collatz(12) == [12, 6, 3, 10, 5, 16, 8, 4, 2, 1]
assert len(collatz(256)) == 9
assert len(collatz(257)) == 123
```

source de l'exercice (http://inventwithpython.com/pythongently/exercise39/)

Bonus

Section *Les générateurs*

Plutôt que de retourner une liste, on pourrait produire un générateur : `gen_collatz(n)`.
L'intérêt est d'économiser de la mémoire, mais il peut être moins évident d'interroger la séquence : valeur maximum ? taille ?

Solution

[45]:
```
def collatz(n):
    assert isinstance(n, int)
    assert n > 0
    sequence = [n]
    while n != 1:
        is_even = n%2==0
        if is_even:
            n //= 2
        else:
            n = 3*n+1
        sequence.append(n)
    return sequence
```

[46]:
```
assert collatz(1) == [1]
assert collatz(10) == [10, 5, 16, 8, 4, 2, 1]
```

(suite sur la page suivante)

(suite de la page précédente)

```
assert collatz(11) == [11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1]
assert collatz(12) == [12, 6, 3, 10, 5, 16, 8, 4, 2, 1]
assert len(collatz(256)) == 9
assert len(collatz(257)) == 123
```

Solution (bonus)

[2]:
```
def gen_collatz(n):
    assert isinstance(n, int)
    assert n > 0
    yield n
    while n != 1:
        is_even = n%2==0
        if is_even:
            n //= 2
        else:
            n = 3*n+1
        yield n
```

[4]:
```
assert list(gen_collatz(1)) == [1]
assert list(gen_collatz(10)) == [10, 5, 16, 8, 4, 2, 1]
assert list(gen_collatz(11)) == [11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2,␣
↪1]
assert list(gen_collatz(12)) == [12, 6, 3, 10, 5, 16, 8, 4, 2, 1]
assert len(list(gen_collatz(256))) == 9
assert len(list(gen_collatz(257))) == 123
assert max(gen_collatz(10)) == 16
assert max(gen_collatz(11)) == 52
assert max(gen_collatz(12)) == 16
```

6.1.7 Exercice 7 : statistiques d'un échantillon

Énoncé

Section *Lire et écrire des fichiers*

Créer une fonction `get_stats(fname)` qui renvoie les minimum, maximum, moyenne et écart-type d'une liste de nombres lus dans un fichier de nom `fname`.

Le résultat se fera sous la forme d'un dictionnaire de clés `'min'`, `'max'`, `'mean'`, `'std'`.

NB :

— les fonctions `min`, `max`, `mean`, `std` peuvent être importées depuis le module numpy
— on admettra que le fichier d'entrée n'a qu'une seule valeur par ligne, du type :

```
3548.279704
3589.7804263
3095.09156402
...
```

Vérifier les résultats :

```
assert get_stats('demo_stats.txt') # s'assure que la fonction ne retourne pas un␣
↪dictionnaire vide
assert get_stats('demo_stats.txt')['min'] == 2561.50070722
assert get_stats('demo_stats.txt')['max'] == 3722.25284478
assert np.allclose(get_stats('demo_stats.txt')['mean'], 3150)
assert np.allclose(get_stats('demo_stats.txt')['std'], 314)
```

Solution

[54]:
```
import numpy as np

def get_stats(fname):
    values = []
    with open(fname) as rd:
        for line in rd:
            values.append(float(line.strip()))
    return dict(
        min=np.min(values),
        max=np.max(values),
        mean=np.mean(values),
        std=np.std(values)
    )
```

[66]:
```
assert get_stats('demo_stats.txt') # s'assure que la fonction ne retourne pas un␣
↪dictionnaire vide
assert get_stats('demo_stats.txt')['min'] == 2561.50070722
assert get_stats('demo_stats.txt')['max'] == 3722.25284478
assert np.allclose(get_stats('demo_stats.txt')['mean'], 3150)
assert np.allclose(get_stats('demo_stats.txt')['std'], 314)
```

Bonus

Ne pas réinventer la roue, des librairies font très bien le travail !

[67]:
```
import pandas as pd
values = pd.read_csv('demo_stats.txt', names=['Sample'])
values.describe()
```

[67]:
```
            Sample
count    50.000000
mean   3150.000000
std     317.187899
min    2561.500707
25%    2950.424439
50%    3160.149710
75%    3414.406663
max    3722.252845
```

6.1.8 Exercice 8 : filtrage de liste

Énoncé

Section *Listes en intension*

Créer une fonction `keep_positive(values)` qui recoit une liste de valeurs et renvoie une liste similaire mais avec seulement les valeurs positives.

Vérifier les résultats :

```
assert keep_positive([4, 5, 2, 9]) == [4, 5, 2, 9]
assert keep_positive([1, -2, 3, -4]) == [1, 3]
assert keep_positive([-1, -2]) == []
```

Solution

[8]:
```
keep_positive = lambda values: [v for v in values if v>=0]
```

[9]:
```
assert keep_positive([4, 5, 2, 9]) == [4, 5, 2, 9]
assert keep_positive([1, -2, 3, -4]) == [1, 3]
assert keep_positive([-1, -2]) == []
```

6.1.9 Exercice 9 : matrice de zéros

Enoncé

Section *Noms et objets non immuables*

Créer une fonction `make_zeros_matrix(nrows, ncols)` qui produit une liste de `nrows` listes, chacune remplie de `ncols` zéros.
Cette structure imbriquée est un prototype pour une représentation de matrice, on peut jouer à y affecter des valeurs dans des cellules via une double indexation.

Une idée pourrait être la suivante :

```
# It's a trap !
def make_zeros_matrix(nrows, ncols):
    return [[0] * ncols] * nrows
```

Mais il s'avère que la matrice produite a un problème… lequel ?

Vérifier les résultats :

```
assert make_zeros_matrix(1, 1) == [[0]]
assert make_zeros_matrix(4, 0) == [[], [], [], []]
assert make_zeros_matrix(0, 4) == []
assert make_zeros_matrix(3, 2) == [[0, 0],
                                   [0, 0],
                                   [0, 0]]
mat = make_zeros_matrix(2, 3)
assert mat == [[0, 0, 0],
               [0, 0, 0]]
```

(suite sur la page suivante)

(suite de la page précédente)

```
mat[0][0] = 1
assert mat == [[1, 0, 0],
               [0, 0, 0]]
```

Solution

[20]:
```
def make_zeros_matrix(nrows, ncols):
    matrix = []
    for _ in range(nrows):
        matrix.append([0] * ncols)
    return matrix
```

[21]:
```
assert make_zeros_matrix(1, 1) == [[0]]
assert make_zeros_matrix(4, 0) == [[], [], [], []]
assert make_zeros_matrix(0, 4) == []
assert make_zeros_matrix(3, 2) == [[0, 0],
                                   [0, 0],
                                   [0, 0]]
mat = make_zeros_matrix(2, 3)
assert mat == [[0, 0, 0],
               [0, 0, 0]]
mat[0][0] = 1
assert mat == [[1, 0, 0],
               [0, 0, 0]]
```

Quel est le problème avec la suggestion de l'énoncé ?

La structure imbriquée construite contient en fait UNE SEULE LISTE POUR TOUTES LES LIGNES !!!

Modifier une ligne va modifier cette liste, et donc modifier toutes les lignes…

[2]:
```
def wrong_make_zeros_matrix(nrows, ncols):
    return [[0] * ncols] * nrows

wrong_mat = wrong_make_zeros_matrix(2, 3)
wrong_mat[0][0] = 1
assert wrong_mat == [[1, 0, 0],
                     [1, 0, 0]]  # <- cette ligne ne devrait pas être modifiée !
```

La preuve en images ci-dessous

lien (https://pythontutor.com/render.html#code=def%20wrong_make_zeros_matrix%28nrows,%20ncols%29%3A%0A%20%20%20%20return%20%5B%5B0%5D%20*%20ncols%5D%20*%20nrows%0A%0Awrong_mat%20%3D%20wrong_make_zeros_matrix%282,%203%29%0Awrong_mat%5B0%5D%5B0%5D%20%3D%201&cumulative=false&curInstr=0&heapPrimitives=nevernest&mode=display&origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false)

[1]:
```
from IPython.display import IFrame
IFrame("https://pythontutor.com/iframe-embed.html#code=def%20wrong_make_zeros_matrix
↪%28nrows,%20ncols%29%3A%0A%20%20%20%20return%20%5B%5B0%5D%20*%20ncols%5D%20*%20nrows
↪%0A%0Awrong_mat%20%3D%20wrong_make_zeros_matrix%282,%203%29%0Awrong_mat%5B0%5D%5B0
↪%5D%20%3D%201&cumulative=false&curInstr=0&heapPrimitives=nevernest&mode=display&
```

(suite sur la page suivante)

(suite de la page précédente)

```
↪origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false",
      width='100%', height=300)
```

```
[1]: <IPython.lib.display.IFrame at 0x7fa4d16eb150>
```

6.2 Encore plus d'exercices

— https://inventwithpython.com/pythongently/
42 exercices bien expliqués pour découvrir python
— https://dabeaz-course.github.io/practical-python/
un cours complet, avec pleins d'exercices
— https://nedbatchelder.com/text/kindling.html

« Kindling projects » : plus gros qu'un exercice de cours, plus petit qu'un « vrai » projet
une liste d'idées et de pointeurs vers des listes d'idée, accessibles aux débutants