HE VD HAUTE ÉCOLE D'INGÉNIERIE ET DE GESTION DU CANTON DE VAUD

Hes·so Haute Ecole Spécialisée de Suisse occidentale

Résumé du cours à choix de Python

Département : TIN
Unité d'enseignement : Python

Auteurs : Maillard Arnaud
Date : June 7, 2023

Contents

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import pint
```

1 Structure de données

1.1 Les scalaires

Les différentes structures de données scalaires sont les suivantes :

- Integers
- Floats
- Complex
- Strings
- Boolean
- None

[2]:
```
i = int(42) # Integers
print(type(i))
f = float(42.0) # Floating point numbers
print(type(f))
c = 42.0 + 0.0j # Complex numbers
print(type(c))
b = True # Booleans
print(type(b))
s = "42" # Strings
print(type(s))
n = None # NoneType
print(type(n))
```

```
<class 'int'>
<class 'float'>
<class 'complex'>
<class 'bool'>
<class 'str'>
<class 'NoneType'>
```

1.2 Conteneurs

Il y a différents types de conteneurs en python : - Listes : utilisent le symbole `[]` - `Tuples : utilisent le symbole()- Dictionnaires : utilisent le symbole- Set : utilisent le symbole‘`

1.2.1 Listes

Les principales propriétés des listes sont les suivantes : - Non hashable - Possède un itérateur

Non hashable : on ne peut générer une valeur unique à partir de la liste. On ne peut donc pas utiliser une liste comme clé dans un dictionnaire.

Possède un itérateur : on peut parcourir la liste avec une boucle for.

[3]:
```
l = [1, 2, 3, 4, 5] # Liste

# Non-hashable
try :
    print(hash(l))
except TypeError as e:
    print(e)

# Possède un itérateur
iter_l = iter(l)
print('\nPossède un itérateur :')
print(next(iter_l))
print(next(iter_l))

print('\nBoucle for :')
for i in l:
    print(i)

# Next ne marche pas si l'itérateur n'est pas stocké dans une variable
print('\nNext ne marche pas si l\'itérateur n\'est pas stocké dans une variable :
 ↪')
print(next(iter(l)))
print(next(iter(l)))
```

```
unhashable type: 'list'

Possède un itérateur :
1
2

Boucle for :
1
2
3
4
```

```
5

Next ne marche pas si l'itérateur n'est pas stocké dans une variable :
1
1
```

Compréhension de liste :

[4]:
```python
s = "1 2 3 4 5" # String

print('\nString :')
print(s)

print('Séparation :')
s_sep = [s.strip() for s in s.split(' ')]
print(s_sep)

print('Transformation en int :')
s_int = [int(s) for s in s_sep]
print(s_int)

print('Ajout de 10 :')
s1 = [s_int+ 10 for s_int in [1,2,3,4]]
print(s1)
```

```
String :
1 2 3 4 5
Séparation :
['1', '2', '3', '4', '5']
Transformation en int :
[1, 2, 3, 4, 5]
Ajout de 10 :
[11, 12, 13, 14]
```

1.2.2 Tuples

Les principales propriétés des tuples sont les suivantes : - Hashable - Possède un itérateur - Non modifiable

Hashable : on peut générer une valeur unique à partir du tuple. On peut donc utiliser un tuple comme clé dans un dictionnaire.

Possède un itérateur : on peut parcourir le tuple avec une boucle for.

Non modifiable : on ne peut pas modifier un tuple. On peut seulement le parcourir.

[5]:
```python
# Hashable
t = (1, 2, 3, 4, 5) # Tuple
print('\nHashable :')
try :
    print(hash(t))
except TypeError as e:
    print(e)

# Possède un itérateur
iter_t = iter(t)
print('\nPossède un itérateur :')
print(next(iter_t))
print(next(iter_t))

print('\nBoucle for :')
for i in t:
    print(i)

# Non modifiable
print('\nNon modifiable :')
try :
    t[0] = 42
except TypeError as e:
    print(e)
```

```
Hashable :
-5659871693760987716

Possède un itérateur :
1
2

Boucle for :
1
2
3
4
```

```
5

Non modifiable :
'tuple' object does not support item assignment
```

1.2.3 Dictionnaires

Les principales propriétés des dictionnaires sont les suivantes : - Les clés doivent être hashables (donc pas de listes comme clés) - Lent à parcourir - Propriétés d'un dictionnaire : - .items() : retourne les clés et les valeurs - .values() : retourne les valeurs - .keys() : retourne les clés - Hash Table

Hash Table : un dictionnaire est une table de hashage. Il calcule un hash code pour chaque clé.

[6]:
```python
d = {'a' : 1, 'b' : 2, 'c' : 3} # Dictionnaire hashable
print('Dictionnaire hashable :')
print(d)

# Les clés dovient être hashable
print('\nLes clés doivent être hashable :')
try :
    d_unhashable = {[1, 2, 3] : 1, [4, 5, 6] : 2}
except TypeError as e:
    print(e)

# Propriétés des dictionnaires
print('\nPropriétés des dictionnaires :')
print('Keys :', d.keys())
print('Values :', d.values())
print('Items :', d.items())
```

```
Dictionnaire hashable :
{'a': 1, 'b': 2, 'c': 3}

Les clés doivent être hashable :
unhashable type: 'list'

Propriétés des dictionnaires :
Keys : dict_keys(['a', 'b', 'c'])
Values : dict_values([1, 2, 3])
Items : dict_items([('a', 1), ('b', 2), ('c', 3)])
```

1.2.4 Set

Les principales propriétés des sets sont les suivantes : - Un dictionnaire avec que des clés. - Rend les éléments uniques

[7]:
```
u = {1, 2, 3, 4, 5} # Set
print('\nType :', type(u))

# Rend les éléments uniques
print('\nRend les éléments uniques :')
u = {1,1,1,1,1,2,2,2,3,4,4,5}
print(u)
```

```
Type : <class 'set'>

Rend les éléments uniques :
{1, 2, 3, 4, 5}
```

2 Classes

Python est un langage orienté objet. Il est donc possible de créer des classes.

Voici quelques-unes des méthodes `magiques` des classes python sont les suivantes : - `__new__` : qui est le constructeur de la classe - `__init__` : initialisateur de classe qui peut prendre des paramètres - `__repr__` : qui retourne une représentation de la classe

[8]:
```
class A:
    def __new__(cls, a):
        print('Constructeur')
        return super().__new__(cls)
    def __init__(self, a):
        print('Initialisation')
        self.a = a
    def __repr__(self):
        return str('A : ' + str(self.a))
    def __next__(self):
        self.i += 1
        return self.a[self.i]
    def __iter__(self):
        return self.a

a = A([1, 2, 3, 4, 5])

print(repr(a))
```

```
Constructeur
Initialisation
A : [1, 2, 3, 4, 5]
```

2.1 Propriétés

Il est possible de créer des propriétés dans une classe. Pour cela, il faut utiliser les décorateurs `@property`

[14]:
```python
class Shape:
    def __init__(self, x, y, color):
        self.x = x
        self.y = y
        self._color = color

    def area(self):
        return 42

    def move(self, dx, dy):
        self.x += dx
        self.y += dy

    # On ne fait pas ça en python :
    def getColor(self):
        return self._color

    # On fait ça en python :
    @property
    def color(self):
        return self._color


class Circle(Shape):  # Circle hérite de Shape
    def __init__(self, radius, x, y, color):
        # Appel du constructeur de Shape (parent) avec  le mot clé super()
        super().__init__(x, y, color)
        self._radius = radius

    def _compute_area(self):
        self.area = 3.14 * self.radius**2

    # Getter de radius
    @property
    def radius(self):
        return self._radius

    # Setter de radius
    @radius.setter
    def radius(self, value):
        if value < 0:
            raise ValueError("Radius cannot be negative")
        self._radius = value
```

```
        self._compute_area()
```

3 Ouvrir, lire et fermer des fichiers

Les différentes méthodes utiles à la gestion des fichiers : - `open(filename, 'r')` : ouvre un fichier en lecture - `open(filename, 'w')` : ouvre un fichier en écriture - `read()` : lis tout le fichier - `readline()` : lis la prochaine ligne - `readlines()` : lis toutes les lignes - `close()` : ferme le fichier

[9]:
```
fp = open('resume_semestre.ipynb', 'r') # 'w', 'a'

print(fp.readline()) # Lis la prochaine ligne
print(fp.readline()) # Lis la prochaine ligne

fp.close()
```

```
{

 "cells": [
```

4 Surcharge d'opérateurs

Toutes les surcharges d'opérateurs sont disponibles dans le data model python, mais pour les opérateurs numériques, on peut se référer à la section sur les opérateurs numériques.

4.0.1 Exemple 1

[1]:
```
class Number:
    def __init__(self, number):
        self.number = number
    def __add__(self, n) :
        return self.number + n
    def __sub__(self, n) :
        return self.number - n
    def __mul__(self, n) :
        return self.number * n
    def __floordiv__(self, n) : # //
        return self.number / n
    def __truediv__(self, n): # /
        return self.number / n

n = Number(42)

n - 33
```

[1]:
```
9
```

[2]:
```
n*10
```

[2]: 420

Attention, ici le type retourné est un `int` et non un `Number`.

4.0.2 Exemple 2

[6]:
```
class Number:
   def __init__(self, number):
       self.number = number
   def __add__(self, n) :
       return Number(self.number + n)
   def __repr__(self):
       return str(self.number)

n = Number(42)
n
```

[6]: 42

[7]:
```
Number(42)
```

[7]: 42

Ici, on utilise `__repr__` pour afficher un `Number` comme un `int`.

4.0.3 Exemple 3

[11]:
```
class Number:
    def __init__(self, number):
        self.number = number
    def __add__(self, n) :
        return Number(self.number + n)
    def __repr__(self):
        return f"Fstring : Number({self.number})"

Number(42)
```

[11]: Fstring : Number(42)

On utilise ici une `f-string` pour afficher un `Number` comme on le souhaite.

4.0.4 Exemple 4

[14]:
```
class Number:
    def __init__(self, number):
        self.number = number
    def __add__(self, n) :
        return Number(self.number + n)
    def __repr__(self):
        return f"Number({self.number})"
    def __str__(self) :
        return str(self.number)

print(Number(42))
```

```
42
```

[16]:
```
Number(42)
```

[16]:
```
Number(42)
```

Finalement on peut voir la différence entre `__repr__` et `__str__` ci-dessus.

4.0.5 Exemple 5

[17]:
```
n + 33
```

[17]:
```
Number(75)
```

[19]:
```
try :
    33 + n
except TypeError as e:
    print(e)
```

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

4.0.6 Exemple 6

[30]:
```
class Number:
    def __init__(self, number):
        self.number = number
    def __add__(self, n) :
        print("Add")
        return Number(self.number + n)
    def __radd__(self, n):
        print("Radd")
        return self + n # utilisation de la surcharge déjà faite auparavant
    def __repr__(self):
        return f"Number({self.number})"
    def __str__(self) :
        return str(self.number)

n = Number(42)
```

[31]:
```
n + 33
```

```
Add
```

[31]:
```
Number(75)
```

[32]:
```
33 + n
```

```
Radd
Add
```

[32]:
```
Number(75)
```

Ici on surcharge `__radd__` pour que l'opérateur `+` soit associatif. Afin de simplifier le code on utilise la surcharge déjà faite auparavant dans `__radd__`.

5 Opérateurs

5.0.1 Opérateurs fonctionnels pour tous les conteneurs de données

[33]:
```
{2, 3, 4, 5} - {2,4}
```

[33]:
```
{3, 5}
```

[36]:
```
{2,3,4,5,6}^{2,3,4} # garde uniquement les éléments qui ne sont pas dans les␣
↪deux sets (XOR)
```

[36]:
```
{5, 6}
```

5.0.2 Opérateur `all`

[38]:
```
all((1,2,3,4))
```

[38]:
```
True
```

[40]:
```
all((1,2,3,4,0))
```

[40]:
```
False
```

[46]:
```
any((0,0,0,0,0))
```

[46]:
```
False
```

[48]:
```
any((0,0,0,0,1))
```

[48]:
```
True
```

5.0.3 Swapper

[50]:
```
a = 1
b = 2

a, b = b, a

a
```

[50]:
```
2
```

[51]:
```
b
```

[51]:
```
1
```

5.0.4 Enumerate

[54]:
```
l = [3,2,5,6]

list(enumerate(l))
```

[54]:
```
[(0, 3), (1, 2), (2, 5), (3, 6)]
```

[56]:
```
l = [3,2,5,6]

for i, v in enumerate(l) :
    print(f"i={i}, v={v}")
```

```
i=0, v=3
i=1, v=2
i=2, v=5
i=3, v=6
```

5.0.5 Zip

[58]:
```
firstnames = ['John', 'Jane', 'Jack']
lastnames = ['Doe', 'Doe', 'Black']

list(zip(firstnames, lastnames))
```

[58]:
```
[('John', 'Doe'), ('Jane', 'Doe'), ('Jack', 'Black')]
```

[59]:
```
[' '.join(x) for x in list(zip(firstnames, lastnames))]
```

[59]:
```
['John Doe', 'Jane Doe', 'Jack Black']
```

5.0.6 Opérateur de déréférencement * et **

[62]:
```
def operate(a,b, **kwargs):
    if 'add' in kwargs:
            print(f"{a}+{b}={a+b}")
    if 'sub' in kwargs:
            print(f"{a}-{b}={a-b}")

operate(23, 42)
```

[64]:
```
operate(23, 42, add=True)
```

```
23+42=65
```

[65]:
```
operate(23, 42, add=True, sub=True)
```

```
23+42=65
23-42=-19
```

L'opérateur `*` permet de déréférencer une liste ou un tuple. Il permet de passer les éléments d'une liste ou d'un tuple comme paramètres d'une fonction.

Pour une liste : `l = [1,2,3,4]`, si on donne comme argument `*l`, alors on aura comme argument `1,2,3,4` et donc c'est la liste sans les corchets donc 5 éléments

[78]:
```
l = [1,2,3,4,5]

def toto(*args, **kwargs) :
    print("Args", args)
    print("Kwargs", kwargs)

l
```

[78]:
```
[1, 2, 3, 4, 5]
```

[80]:
```
toto(*l, "salut", 4, *l)
```

```
Args (1, 2, 3, 4, 5, 'salut', 4, 1, 2, 3, 4, 5)
Kwargs {}
```

[81]:
```
toto(*l, "salut", 4, *l, a=1, b=2)
```

```
Args (1, 2, 3, 4, 5, 'salut', 4, 1, 2, 3, 4, 5)
Kwargs {'a': 1, 'b': 2}
```

6 Yield return

Le `yield` permet de retourner une valeur et de mettre la fonction en pause. Lorsque la fonction est appelée à nouveau, elle reprend là où elle s'était arrêtée.

[69]:
```
def foo():
    i = 0
    while True :
        i += 1
        yield i

foo()
```

[69]:
```
<generator object foo at 0x7ffb84050580>
```

[71]:
```
next(foo())
```

[71]:
```
1
```

[73]:
```
next(foo())
```

[73]:
```
1
```

[74]:
```
g = foo()
```

[75]:
```
next(g)
```

[75]:
```
1
```

7 Named Tuple

Les `namedtuple` sont des tuples qui ont des noms. Ils sont donc plus faciles à utiliser que les tuples classiques.

[87]:
```
from collections import namedtuple

Type = namedtuple('TypeName', ('field1', 'field2'))
t = Type('a', 'b')
t[0]
```

[87]:
```
'a'
```

[85]:
```
t.field1
```

[85]:
```
'a'
```

8 Numpy

[97]:
```
import numpy as np
a = np.arange(1, 6) # [1,2,3,4,5]
b = np.ones(5) # [1,1,1,1,1]

a2 = a[:, None]
print('Column vector :\n',a2)
broadcasted = a2 * b
print('Broadcasted :\n',broadcasted)
```

```
Column vector :
 [[1]
 [2]
 [3]
 [4]
 [5]]
Broadcasted :
 [[1. 1. 1. 1. 1.]
 [2. 2. 2. 2. 2.]
 [3. 3. 3. 3. 3.]
 [4. 4. 4. 4. 4.]
 [5. 5. 5. 5. 5.]]
```

9 Software Design Principles

Un principe de conception est une règle générale qui guide le développement d'un logiciel. Il est souvent exprimé sous forme de phrase, et peut être utilisé pour évaluer la qualité d'un logiciel.

- [x] SSOT
- [x] SOLID
- [x] KISS
- [x] DRY/WET
- [x] YAGNI

9.1 SSOT (Single Source of Truth)

La règle du SSOT s'applique aux données et pas à la redondance du code. Elle stipule que les données doivent être stockées dans un seul endroit. Si une donnée doit être modifiée, elle doit être modifiée dans un seul endroit.

Mauvais exemple :

```
equipageDes42MillesEtUneNuits = {
"capitaine" : 42, # Le capitaine a 42 ans
"moussailon" : 42,
"machinininiste" : 42
}
```

Bon exemple :

```
ageEquipage = 42

equipageDes42MillesEtUneNuits = {
"capitaine" : ageEquipage,# Le capitaine a un certain age
"moussailon" : ageEquipage,
"machinininiste" : ageEquipage
}
```

9.1.1 DRY (Don't Repeat Yourself)

Le contraire est le WET (Write Everything Twice).

Exemple de code WET :

```
def calculerSomme(liste):
    somme = 0
    for element in liste:
        somme += element
    return somme

sommePommes = calculerSomme(liste[pommes])
sommePoires = calculerSomme(liste[poires])
sommeBananes = calculerSomme(liste[bananes])
```

Une meilleure solution :

```
sommes = {}
for key, values in liste.items():
    sommes[key] = calculerSomme(values)
```

Et avec une compréhension :

```
sommes = {key: calculerSomme(values) for key, values in liste.items()}
```

9.1.2 KISS (Keep It Simple, Stupid)

Une fonction ça devrait entre 10 lignes et un écran. Si c'est plus compliqué, c'est qu'il y a un problème.

9.1.3 YAGNI (You Ain't Gonna Need It)

Ne pas coder des fonctionnalités qui ne sont pas nécessaires. C'est une bonne règle pour éviter de coder des fonctionnalités qui ne seront jamais utilisées.

Par exemple, le gars qui a construit le moteur d'avion sur l'image ci-dessous n'a pas suivi cette règle. Il a installé une selle dessus parce qu'il se disait que ça pourrait être utile un jour, mais ça n'a jamais été utilisé :

10 Pipenv

C'est pour avoir une sorte de bac à sable où l'on peut installer des packages python sans les installer globalement sur son ordinateur.

10.1 Création d'un environnement virtuel

Pour créer un enivronnement on utilise la commande : `python3 -mvenv venv`

Exemple :

```
  projet git:(master)  python3 -mvenv venv
  projet git:(master)  ls -a
.  ..  __init__.py  __main__.py  venv
  projet git:(master)  ls -la
total 0
drwxrwxrwx 1 arnaud-maillard1 arnaud-maillard1 4096 Apr 18 11:15 .
drwxrwxrwx 1 arnaud-maillard1 arnaud-maillard1 4096 Apr 18 11:04 ..
-rwxrwxrwx 1 arnaud-maillard1 arnaud-maillard1   44 Apr 18 11:06 __init__.py
-rwxrwxrwx 1 arnaud-maillard1 arnaud-maillard1   35 Apr 18 11:08 __main__.py
drwxrwxrwx 1 arnaud-maillard1 arnaud-maillard1 4096 Apr 18 11:19 venv
```

10.2 Activation de l'environnement virtuel

Pour activer l'environnement virtuel, on utilise la commande : `source venv/bin/activate`

Exemple :

```
  projet git:(master)  source venv/bin/activate
(venv)   projet git:(master)  which python3
/mnt/c/Users/arnau/OneDrive/Documents/HEIGVD/Semestre_6/Python/diary_perso_python/8/projet/venv/
(venv)   projet git:(master)  deactivate
  projet git:(master)  which python3
/usr/bin/python3
  projet git:(master)
```

10.3 Installation de packages

Pour installer un package, on utilise la commande : `pip install <package>` lorsqu'on a déjà activé l'environnement

Exemple :

```
(venv)   projet git:(master)  pip install autopep8
Collecting autopep8
  Downloading autopep8-2.0.2-py2.py3-none-any.whl (45 kB)
     || 45 kB 944 kB/s
Collecting pycodestyle>=2.10.0
  Using cached pycodestyle-2.10.0-py2.py3-none-any.whl (41 kB)
Collecting tomli; python_version < "3.11"
  Using cached tomli-2.0.1-py3-none-any.whl (12 kB)
Installing collected packages: pycodestyle, tomli, autopep8
Successfully installed autopep8-2.0.2 pycodestyle-2.10.0 tomli-2.0.1
(venv)   projet git:(master)  pip list
Package       Version
------------- -------
autopep8      2.0.2
numpy         1.24.2
pip           20.0.2
pkg-resources 0.0.0
pycodestyle   2.10.0
scipy         1.10.1
setuptools    44.0.0
tomli         2.0.1
(venv)   projet git:(master)
```

11 Pint

C'est une librairie python qui permet de faire des conversions de unités. Par exemple :

[98]:
```
from pint import UnitRegistry
ureg = UnitRegistry()
distance = 10 * ureg.kilometer
print(distance.to(ureg.meter))
```

```
10000.0 meter
```

12 Debugger python

On a une erreur dans notre code et on aimerait debugger :

```
(venv)   projet git:(master)  python3 -mhello
Hello world !
Traceback (most recent call last):
  File "/usr/lib/python3.8/runpy.py", line 194, in _run_module_as_main
    return _run_code(code, main_globals, None,
  File "/usr/lib/python3.8/runpy.py", line 87, in _run_code
    exec(code, run_globals)
  File "/mnt/c/Users/arnau/OneDrive/Documents/HEIGVD/Semestre_6/Python/diary_perso_python/8/proj
    sayHello()
  File "/mnt/c/Users/arnau/OneDrive/Documents/HEIGVD/Semestre_6/Python/diary_perso_python/8/proj
    i = i/0
ZeroDivisionError: division by zero
(venv)   projet git:(master)  ipython
/home/arnaud-maillard1/.local/lib/python3.8/site-packages/IPython/core/interactiveshell.py:882:
  warn(
Python 3.8.10 (default, Mar 13 2023, 10:26:41)
Type 'copyright', 'credits' or 'license' for more information
IPython 8.6.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import hello

In [2]: hello.sayHello
Out[2]: <function hello.sayHello()>

In [3]: hello.sayHello()
Hello world !
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Cell In [3], line 1
----> 1 hello.sayHello()

File /mnt/c/Users/arnau/OneDrive/Documents/HEIGVD/Semestre_6/Python/diary_perso_python/8/projet/
      5 i = i+1
      6 i = i*10
```

```
----> 7 i = i/0

ZeroDivisionError: division by zero

In [4]: %debug
> /mnt/c/Users/arnau/OneDrive/Documents/HEIGVD/Semestre_6/Python/diary_perso_python/8/projet/hel
      3     print("Hello world !")
      4     i = 3
      5     i = i+1
      6     i = i*10
----> 7     i = i/0

ipdb> u
> <ipython-input-3-42affdb2d76d>(1)<module>()
----> 1 hello.sayHello()

ipdb> d
> /mnt/c/Users/arnau/OneDrive/Documents/HEIGVD/Semestre_6/Python/diary_perso_python/8/projet/hel
      3     print("Hello world !")
      4     i = 3
      5     i = i+1
      6     i = i*10
----> 7     i = i/0
```

On peut naviguer dans le debug avec les commandes suivantes :

- `u` pour remonter dans le code
- `d` pour descendre dans le code
- `n` pour aller à la ligne suivante
- `c` pour continuer l'exécution du code
- `q` pour quitter le debug
- `h` pour afficher l'aide

On peut même faire des commandes python dans le debug :

```
In [5]: %debug
> /mnt/c/Users/arnau/OneDrive/Documents/HEIGVD/Semestre_6/Python/diary_perso_python/8/projet/hel
      3     print("Hello world !")
      4     i = 3
      5     i = i+1
      6     i = i*10
----> 7     i = i/0
ipdb> print(i)
40
ipdb> i = 50
ipdb> print(i)
50
ipdb>
```

13 Pandas

C'est une librairie python qui permet de faire des manipulations de données. Un tutoriel est disponible : LIEN DU TUTORIEL

14 Itertools

Fonctions pour créer des itérateurs efficaces.

- `product` : produit cartésien
- `permutations` : permutations
- `combinations` : combinaisons
- `combinations_with_replacement` : combinaisons avec remplacement

15 Functools

Pour calculer le n-ième nombre de Fibonacci, on peut utiliser la formule suivante :

[7]:
```
import time


def fib(n):
    if n <= 2 : return 1
    return fib(n-1) + fib(n-2)


st = time.time()


fib(20)


et = time.time()
elapsed_time = et - st
print('Execution time:', elapsed_time, 'seconds')
```

```
Execution time: 0.0019948482513427734 seconds
```

Ce qui nous donne un algorithme récursif pas très efficace qui recalculera plusieurs fois les mêmes valeurs.

On peut améliorer avec un cache créé à la volée :

[8]:
```
def fib(n, cache={}):
    if n in cache:
        return cache[n]
    if n < 2:
        return n
    cache[n] = fib(n-1) + fib(n-2)
    return cache[n]

st = time.time()


fib(20)



et = time.time()
elapsed_time = et - st
print('Execution time:', elapsed_time, 'seconds')
```

```
Execution time: 5.078315734863281e-05 seconds
```

Ce qui est déjà mieux, mais on peut faire encore mieux avec le décorateur : `functools.cache` :

[9]:
```
import functools

@functools.lru_cache
def fib(n):
    if n < 2:
        return n
    return fib(n-1) + fib(n-2)

st = time.time()


fib(20)



et = time.time()
elapsed_time = et - st
print('Execution time:', elapsed_time, 'seconds')
```

```
Execution time: 9.322166442871094e-05 seconds
```

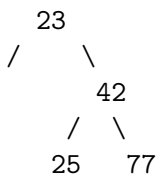
La fonction `lru_cache` fait donc ce qui s'appelle de la *mémoïsation*ou de la programmation dynamique.

16 Heapq

[13]:
```python
import heapq

heap = []
heapq.heappush(heap, 42)
print('Heap 1 :', heap)
heapq.heappush(heap, 23)
print('Heap 2 :', heap)
heapq.heappush(heap, 77)
print('Heap 3 :', heap)
heapq.heappush(heap, 25)
print('Heap 4 :', heap)
```

```
Heap 1 : [42]
Heap 2 : [23, 42]
Heap 3 : [23, 42, 77]
Heap 4 : [23, 25, 77, 42]
```

L'ordre vu comme ceci n'est pas compréhensible, mais si on le voit comme **un arbre binaire**, on comprend mieux :



Tous les nombres à droite de 23 seront plus grands que 23, et tous les nombres à gauche de 23 seront plus petits que 23.

17 Flask

Flask est un framework python pour faire des sites web. Il est très simple à utiliser et très léger.

Un exemple avec une arborescence de fichiers comme ceci :

```
|__ replace_flask.py
|__ market.html
|__ templates
    |__ market_template.html
```

Et le code de `replace_flask.py` :

```python
#!/usr/bin/env python3

from flask import Flask, render_template


app = Flask(__name__)


articles = [
    {'name': 'ballon', 'variantes': [
        {'name': 'foot', 'price': 10}, {'name': 'volley', 'price': 15}]},
    {'name': 'raquette', 'variantes': [
        {'name': 'tennis', 'price': 20}, {'name': 'badminton', 'price': 25}]},
    {'name': 'chaussures', 'variantes': [
        {'name': 'rouge', 'price': 30}, {'name': 'bleu', 'price': 35}, {'name': 'vert', 'price':
]

# Render the template
@app.route('/')
def index():
    return render_template('market_template.html', marketName="Loulou sport", articles=articles)


if __name__ == '__main__':
    app.run(debug=True)
```

Et le code de `market_template.html` :

```
<html>

<head>
<h1>Bienvenue chez {{marketName}}</h1>
</head>

<body>

Voici les articles disponibles dans notre magasin : <br>

<ul>
{% for article in articles %}
    <li>{{article.name}}
        <ul>
        {% for variante in article.variantes%}
            <li>{{variante.name}}, prix : {{variante.price}}</li>
        {% endfor %}
        </ul>
    </li>
{% endfor %}
</ul>

</body>
</html>
```

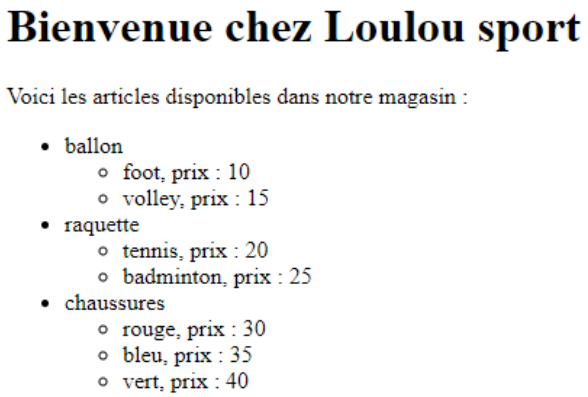
Et le `market.html` :

```
<html>

<head>
<h1>Bienvenue chez Loulou Sport</h1>
</head>

<body>
```

```
Voici les articles disponibles dans notre magasin : <br>

<ul>
    <li>ballon
        <ul>
            <li>foot, prix : 10</li>
            <li>volley, prix : 15</li>
        </ul>
    </li>

    <li>raquette
        <ul>
            <li>tennis, prix : 20</li>
            <li>badminton, prix : 25</li>
        </ul>
    </li>

    <li>chaussures
        <ul>
            <li>rouge, prix : 30</li>
            <li>bleu, prix : 35</li>
            <li>vert, prix : 40</li>
        </ul>
    </li>
</ul>

</body>
</html>
```

On le lance avec la commande `python3 replace_flask.py` et ensuite, on clique sur le lien qui s'affiche dans le terminal.

On obtient alors le résultat suivant :



Bienvenue chez Loulou sport

Voici les articles disponibles dans notre magasin :

- ballon
  - foot, prix : 10
  - volley, prix : 15
- raquette
  - tennis, prix : 20
  - badminton, prix : 25
- chaussures
  - rouge, prix : 30
  - bleu, prix : 35
  - vert, prix : 40

18 Singleton

Le design pattern Singleton permet de créer une classe qui n'a qu'une seule instance. C'est un design pattern très utilisé pour les loggers.

[16]:
```python
# Singleton pattern (single instance of a class)

# Manière 1 (Pas top)
from typing import Any


class Singleton(object):
    _instance = None

    def __new__(cls, *args, **kwargs):
        if not cls._instance:
            print("Creating instance")
            cls._instance = super().__new__(cls, *args, **kwargs)
        return cls._instance

# Manière 2 (Pythonic)
class Singleton2(object):
    _instances = {}

    def __call__(cls, *args, **kwargs):
        if cls not in cls._instances:
            cls._instances[cls] = super().__call__(*args, **kwargs)
        return cls._instances[cls]


class Logger(object):
    __metaclass__ = Singleton2
```

19 Création d'un package à publié sur PyPi

19.1 Fichier spéciaux

- `.editorconfig` : pour définir les règles de codage
- `.gitignore` : pour ignorer les fichiers à ne pas mettre sur git
- `README.md` : pour décrire le package
- `setup.py` : fichier utilisé par pip pour qu'il installe le package
- `setup.cfg` : fichier de configuration pour le package
- `pyproject.toml` : fichier de configuration pour le package

19.2 Commandes

- `pip install -e .` : pour installer le package en mode développement en se trouvant dans le dossier du package

Si une erreur mentionne : *LookupError: setuptools-scm was unable to detect version for /mnt/c/Users/arnau/OneDrive/Documents/HEIGVD/Semestre_6/Python/diary_perso_python/13/heigvd*

Il faut faire créer un repo git et faire une version avec un tag.

1. `git init`
2. `git add .`
3. `git commit -m "Initial commit"`
4. `git tag -a 0.0.1`
5. `git describe` -> doit fonctionner

20 Exceptions

Les exceptions en python permettent de simplifier le code

[22]:
```
try :
    a = 12/0
except :
    print("Division par 0")
```

```
Division par 0
```

20.1 Exemple 1

[23]:
```
codes = [4,8,15,16,23,42]

def get_code(i) :
    if (i > len(codes)) :
        raise ValueError('Mauvais valeur de i')
    return codes[i]
```

Mais c'est mieux de faire ceci, car aucun test n'est nécessaire :

[24]:
```
def get_code(i) :
    try :
        return codes[i]
    except IndexError :
        raise ValueError('Mauvais valeur de i')
```

21 Les décorateurs

Les décorateurs permettent d'ajouter des fonctionnalités à une fonction sans la modifier.

[25]:
```
def my_decorator(func):
    def wrapper(*args, **kwargs):
        print('Before')
        func(*args, **kwargs)
        print('After')
    return wrapper

@my_decorator
def foo():
    print('Inside')

foo()
```

```
Before
Inside
After
```