Computer science

Chapter 1: GitHub

How to upload on GitHub

1. Open git bash
2. $ git config -- global user . name " Sébastien Combéfis "
3. $ git config -- global user . email " seb478@gmail . com "
4. Link your directory from the github repository you've created
   git clone https :// github . com / < votreNom >/ AdvancedPython2BA - Labo1 .git
5. git add file.py
6. git commit -m"description"
7. git push

Chapter 2: regex

- Import re
- Pattern = r'[0-9]{9}' =>patterne qu'on recherche
- P = re.compile(pattern)
- Print(p.match("789464341")) => if not correct it will return None

Description de motif (1)

- Précéder les méta-caractères avec un backslash

  `. ^ $ * + ? { } [ ] \ | ( )`

- Classes de caractères définies avec les `[ ]`

  `[abc]` : a, b ou c
  `[0-9]` : n'importe quoi entre 0 et 9
  `[a-zA-Z]` : n'importe quoi entre a et z ou entre A et Z
  `[^aeiou]` : n'importe quoi sauf a, e, i, o ou u
  `[a-z&&[^b]]` : intersection entre deux ensembles

- Classes prédéfinies de caractères

  `.` : n'importe quel caractère (sauf retour à la ligne)
  `\d` : un chiffre (équivalent à `[0-9]`)
  `\s` : un caractère blanc (équivalent à `[ \t\n\r\f\v]`)
  `\w` : un caractère alpha-numérique (équivalent à `[a-zA-Z0-9_]`)

- La méthode `match` renvoie un objet du match

  Plusieurs méthodes pour obtenir des informations sur le match

Méthode	Description
`group`	Renvoie la sous-chaine matchée
`start`	Renvoie la position du début du match
`end`	Renvoie la position de la fin du match
`span`	Renvoie un tuple avec les positions de début et de fin du match

- `search` cherche un motif dans une chaine

  Recherche d'une sous-chaine qui matche le motif

- `findall` recherche toutes les sous-chaines qui matchent

  Renvoie une liste des sous-chaines matchées

- `finditer` renvoie un itérateur d'objets de match

  On peut ainsi parcourir les matchs avec une boucle `for`

  - Possibilité de nommer un groupe avec les options Python

    Le groupe est récupéré avec son nom plutôt que sa position

```
pattern = r'^(?P<pseudo>[a-z]+)@(?P<domain>[a-z]+)\.(?P<extension>[
a-z]{2,3})$'
p = re.compile(pattern)

m = p.match('email@example.net')
if m is not None:
    print(m.groups())
    print(m.group(1))
    print(m.group('domain'))
```

```
('email', 'example', 'net')
email
example
```

Examples from the lab 2 :

```
# Exo 1


tel = r'[0][1-9]\d{6}' #begin with a 0 then next number is not 0 then 6 numbers
nbint = r'-?[1-9]\d*'   #with optional minus no 0 at the start

def phone():
    p = re.compile(tel)
    phone = input("enter your phone number (for ex : 04798695) :")
    print(p.match(phone))
    if p.match(phone) == None :
        recall()
def recall():
    phone()
def entier():
    p = re.compile(nbint)
    number = "-7564644654"
    print(p.match(number))

# recall()
# entier()
```

```python
#Exo 2

numbers = r'\d+'
def findnumbers():
    p = re.compile(numbers)
    with open("labo 2 texte.txt") as file:
        for a, line in enumerate(file):
            if len(p.findall(line)):
                print( "line ",a+1, ": ", p.findall(line))
# findnumbers()
```

```python
#Exo 3


url = "https://www.google.com/dio"
lien = r'(?P<protocol>[^./:]+)://(?P<domain>[^/]+)(?:/(?P<path>.*))?'

def decompose():
    u = re.compile(lien)
    groups = ["protocol","domain","path"]
    result = u.match(url)
    for group in groups:
        if(result.group(group)) is not None :
            print(group.capitalize(), result.group(group))
# decompose()
```

```python
# mini projet : mots croisés
lines = [
    r'(EC|CD)[ABS]',
    r'.[GROS]+',
    r'C?[KS]+'
]
columns = [
    r'[^CBF](MC|XD)',
    r'[CRI]*[ACK]',
    r'[AEIOU]*S'
]
answers = [
    ["E","C","A"],      #Line0[col0, col1, col2]
    ["M","R","O"],      #line1[col0 etc]
    ["C","K","S"]
]

separator = ''
def checkregexcrossword(linesregex, columnregex, answer):
    for a, l in enumerate(linesregex) :
        p = re.compile(l)
        # print("at line",a,":",l,"and answer is :",separator.join(answer[a]),"result :",p.match(separator.join(answer[a])))
        if (p.match(separator.join(answer[a]))) is not None:
            for b, c in enumerate(columnregex) :
                word = [list(row) for row in zip(*reversed(answer))]
                for rev in word:
                    rev.reverse()

                q = re.compile(c)


                print("at line",a,":",l,"and column",b,":",c,"answer :",answer[a][b],"result :",p.match(separator.join(answer[a])),"hello there",q.match(separator.join(word[b])))
        else :
            print("not working fo line", a)

checkregexcrossword(lines, columns, answers)
```

```
DEBUG CONSOLE  PROBLEMS  OUTPUT  TERMINAL                                                    2: Python

nisam@Nico-SFFPC MINGW64 ~/Documents/ECAM/info/Q2/Exam-preparation (master)
$ C:/Users/nisam/AppData/Local/Programs/Python/Python37/python.exe "c:/Users/nisam/Documents/ECAM/info/Q2/Exam-preparation/labo 2.py"
at line 0 : (EC|CD)[ABS] and column 0 : [^CBF](MC|XD) answer : E result : <re.Match object; span=(0, 3), match='ECA'> hello there <re.Match object; span=(0, 3), match='EMC'>
at line 0 : (EC|CD)[ABS] and column 1 : [CRI]*[ACK] answer : C result : <re.Match object; span=(0, 3), match='ECA'> hello there <re.Match object; span=(0, 3), match='CRK'>
at line 0 : (EC|CD)[ABS] and column 2 : [AEIOU]*S answer : A result : <re.Match object; span=(0, 3), match='ECA'> hello there <re.Match object; span=(0, 3), match='AOS'>
at line 1 : .[GROS]+ and column 0 : [^CBF](MC|XD) answer : M result : <re.Match object; span=(0, 3), match='MRO'> hello there <re.Match object; span=(0, 3), match='EMC'>
at line 1 : .[GROS]+ and column 1 : [CRI]*[ACK] answer : R result : <re.Match object; span=(0, 3), match='MRO'> hello there <re.Match object; span=(0, 3), match='CRK'>
at line 1 : .[GROS]+ and column 2 : [AEIOU]*S answer : O result : <re.Match object; span=(0, 3), match='MRO'> hello there <re.Match object; span=(0, 3), match='AOS'>
at line 2 : C?[KS]+ and column 0 : [^CBF](MC|XD) answer : C result : <re.Match object; span=(0, 3), match='CKS'> hello there <re.Match object; span=(0, 3), match='EMC'>
at line 2 : C?[KS]+ and column 1 : [CRI]*[ACK] answer : K result : <re.Match object; span=(0, 3), match='CKS'> hello there <re.Match object; span=(0, 3), match='CRK'>
at line 2 : C?[KS]+ and column 2 : [AEIOU]*S answer : S result : <re.Match object; span=(0, 3), match='CKS'> hello there <re.Match object; span=(0, 3), match='AOS'>
```

Chapter 3 & 4: Functional computing and decorators

Functional computing

- Récupération des paramètres positionnels avec `*args`

  La valeur est un tuple reprenant les paramètres dans l'ordre

```
def f(*args):
    print(args)

f(32)
f('12', 98)
```

```
(32,)
('12', 98)
```

- Récupération des paramètres optionnels avec `**kwargs`

  La valeur est un dictionnaire reprenant les paramètres

```
def f(*args, **kwargs):
    print(args)
    print(kwargs)

f(name=99)
f(0, ready=True)
```

```
()
{'name': 99}
(0,)
{'ready': True}
```

- Application d'une fonction à tous les éléments d'une séquence

  map calcule une nouvelle séquence et la renvoie

- Résultat renvoyé sous la forme d'un générateur

  Utilisation de la fonction `list` pour le convertir

```
data = [1, 2, 3]
result = map(lambda x : x ** 2, data)
print(list(result))
```

```
[1, 4, 9]
```

```
data = [1, -2, 0, 7, -3]
result = filter(lambda x : x >= 0, data)
print(list(result))
```

```
[1, 0, 7]
```

```
data = [2+1j, -1j, 4]
print(sum(data))
```

```
(6+0j)
```

Instances callables

- Une instance d'une classe peut être callable

  Il suffit de redéfinir la méthode `__call__`

- Utile pour stocker de l'information entre deux appels

```
1  class Die:
2      def __init__(self):
3          self.roll()
4
5      @property
6      def value(self):
7          return self.__value
8
9      def roll(self):
10         self.__value = random.randint(1, 6)
11
12     def __call__(self):
13         self.roll()
14         return self.value
15
16 die = Die()
17 print(die.value)
18 print(die())
```

Calcul incrémental d'une moyenne (2)

- Mot réservé `nonlocal` pour accéder aux variables libres

  Seulement nécessaire en cas d'accès en modification à la variable

```
1  def make_averager():
2      total = 0
3      count = 0
4      def averager(value):
5          nonlocal total, count
6          total += value
7          count += 1
8          return total / count
9      return averager
10
11 avg = make_averager()
12 print(avg(5))
13 print(avg(7))
14 print(avg(6))
```

Ajouter un comportement à une fonction

- Fonction `decorate` qui prend une fonction en paramètre

  Ajout d'un comportement avant et après l'appel d'une fonction

```
1  def decorate(f):
2      def wrapper():
3          print('Avant appel')
4          f()
5          print('Après appel')
6      return wrapper
7
8  def sayhello():
9      print('Hello!')
10
11 g = decorate(sayhello)
12 g()
```

```
Avant appel
Hello!
Après appel
```

- Une décoration est ajoutée explicitement à une fonction

  Se note avec @ suivi du nom de la fonction décorante

- Un sucre syntaxique est une notation simplificatrice

  sayhello = decorate(sayhello)

```
1  def decorate(f):
2      def wrapper():
3          print('Avant appel')
4          f()
5          print('Après appel')
6      return wrapper
7
8  @decorate
9  def sayhello():
10     print('Hello!')
11
12 sayhello()
```

- Redéfinition de la méthode `__getitem__`

  Elle reçoit un indice en paramètre et renvoie la valeur associée

- Redéfinition de `__len__` pour définir la taille

```
1  import re
2
3  RE_WORDS = re.compile('\w+')
4
5  class Sentence:
6      def __init__(self, text):
7          self.__text = text
8          self.__words = RE_WORDS.findall(text)
9
10     def __getitem__(self, i):
11         return self.__words[i]
12
13     def __len__(self):
14         return len(self.__words)
```

Itérateur vs Itérable

- Une collection est itérable si on sait la parcourir

  Il suffit d'avoir redéfini la méthode `__getitem__`

- Python utilise un itérateur pour parcourir une collection

  Utilisation des fonctions `iter` et `next`

```
1  data = [1, 8, -2, 4]
2  it = iter(data)
3  while True:
4      try:
5          print(next(it), end=' ')
6      except StopIteration:
7          del it
8          break
```

```
1 8 -2 4
```

Génération fainéante

- Génération des factorielles de tous les nombres naturels

  Utilisation de `yield` pour les générer à la demande

```
1  def fact():
2      n = 0
3      result = 1
4      while True:
5          yield (n, result)
6          n += 1
7          result *= n
8
9  genfact = fact()
10 for i in range(5):
11     print(next(genfact), end=' ')
```

```
(0, 1) (1, 1) (2, 2) (3, 6) (4, 24)
```

Examples:

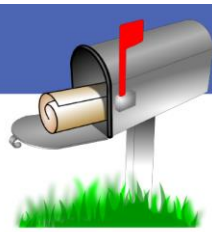
```python
#Exo 1 call hello

def hello():
    print("hello")
def call(func):
    func()
# call(hello)


#Exo 2 call + params

def add(a,b):
    return a+b
def newcall(n, i, j):
    return n(i,j)
# print(newcall(add, 2, 9))


#exo 3 compute

def newadd (a, b):      #DO NOT FORGET THE RETURN
    return a + b
def sub (a, b):
    return a - b
def compute(a, b, op= newadd):
    return op(a,b)
def newnewcall(n, a, b, *args, **kwargs):
    return n (a, b, *args, **kwargs)

# print ( newnewcall ( compute , 2, 9)) # Affiche '11'
# print ( newnewcall ( compute , 2, 9, op= sub ))
```

```python
#Exo 1 decorator

def sleep(t):
    def decorator(f):
        def wrapper(*args,**kwargs):
            res = f(*args,**kwargs)
            time.sleep(t)
            return res
        return wrapper
    return decorator

@sleep(0.1)
def printnum (i):
    print (i)
cnt = 3


# while cnt > 0:
#     printnum (cnt)
#     cnt -= 1
# print ("KA - BOOM !")


#Exo 2


def binrep(n):
    while n>0:
        bit = n% 2
        n //=2
        yield bit

b = binrep(12) # en binaire c'est 1100  or we can  use bin(num)
for i in b:
    print(i)
while True:
    try:
        print(next(b))
    except StopIteration:
        break
```

Chapter 5&6: Network computing

Network communication:

Protocole UDP



- Communication par échange de datagrammes

  Paquets discrets de données

- Caractéristiques du User Datagram Protocol
  - Transfert non fiable (réception et ordre non garantis)
  - Sans connexion (protocole léger)
  - Adapté à l'architecture peer-to-peer
- Identification avec une adresse UDP $\langle IP, Port \rangle$

Protocole TCP

- Communication par flux (stream)

  Flux continu de données

- Caractéristiques du Transmission Control Protocol
  - Transfert fiable (réception et ordre garantis)
  - Avec connexion (protocole lourd)
  - Adapté à l'architecture client/serveur
- Identification avec une adresse TCP $\langle IP, Port \rangle$

Import module "socket"

```
print(socket.getfqdn('www.google.be'))          # Fully Qualified Domain Name
print(socket.gethostname())                     # Nom d'hôte de la machine

print(socket.gethostbyname('www.google.be'))    # Hôte à partir du nom
print(socket.gethostbyaddr('213.186.33.2'))     # Hôte à partir de l'adresse
```

```
wa-in-f94.1e100.net
MacBook-Pro-de-Sebastien-3.local

64.233.184.94
('cluster002.ovh.net', ['2.33.186.213.in-addr.arpa'],
['213.186.33.2'])
```

Création d'un socket

- Socket représenté par un objet de la classe `socket.socket`

  Même classe qui gère tous les types de socket

- Deux paramètres essentiels
  - Famille des adresses (`family`) (`AF_INET` par défaut)
  - Type de socket (`type`) (`SOCK_STREAM` par défaut)

```
1 s = socket.socket()
2 print(s.getsockname())
3
4 t = socket.socket(socket.AF_INET6, socket.SOCK_DGRAM)
5 print(t.getsockname())
```

```
('0.0.0.0', 0)
('::', 0, 0, 0)
```

```
s.connect(('www.python.org', 80))
print(s.getsockname())
```

```
('192.168.1.3', 61774)
```

Binding

- Attacher le socket avec la méthode `bind`

  Associer un socket à une adresse spécifiée

- Adresse socket IPv4 formée du nom de l'hôte et du port

  Adresse représentée par un tuple à deux éléments

```
1 s.bind((socket.gethostname(), 6000))
2 print(s.getsockname())
```

```
('192.168.1.3', 6000)
```

- Utilisation de la méthode `close`

  Doit être fait pour chaque socket ouvert

```
1 s.close()
```

Chat application

1. Create a socket:

```
s = socket.socket(type=socket.SOCK_DGRAM)

data = "Hello World!".encode()
sent = s.sendto(data, ('localhost', 5000))
if sent == len(data):
    print("Envoi complet")
```

Or with a loop:

```
s = socket.socket(type=socket.SOCK_DGRAM)

address = ('localhost', 5000)
message = "Hello World!".encode()

totalsent = 0
while totalsent < len(message):
    sent = s.sendto(message[totalsent:], address)
    totalsent += sent
```

2. Reception:

```
s = socket.socket(type=socket.SOCK_DGRAM)
s.bind((socket.gethostname(), 5000))

data = s.recvfrom(512).decode()
print('Reçu', len(data), 'octets :')
print(data)
```

petite valeur puissance de 2 comme 4096

3. Chatting:

```
class Chat():
    def __init__(self, host=socket.gethostname(), port=5000):
        s = socket.socket(type=socket.SOCK_DGRAM)
        s.settimeout(0.5)
        s.bind((host, port))
        self.__s = s
```

pour que recvfrom soit non bloquant

```
def _exit(self):
    self.__running = False
    self.__address = None
    self.__s.close()

def _quit(self):
    self.__address = None

def _join(self, param):
    tokens = param.split(' ')
    if len(tokens) == 2:
        self.__address = (socket.gethostbyaddr(tokens[0])[0], int(tokens[1]))

def _send(self, param):
    if self.__address is not None:
        message = param.encode()
        totalsent = 0
        while totalsent < len(message):
            sent = self.__s.sendto(message[totalsent:], self.__address)
            totalsent += sent
```

Une méthode pour chaque commande disponible

4. Receive the message:

```
def _receive(self):
    while self.__running:
        try:
            data, address = self.__s.recvfrom(1024)
            print(data.decode())
        except socket.timeout:
            pass
```

évite l'appel bloquant

5. Listening:

```
def run(self):
    handlers = {
        '/exit': self._exit,
        '/quit': self._quit,
        '/join': self._join,
        '/send': self._send
    }
    self.__running = True
    self.__address = None
    threading.Thread(target=self._receive).start()
    # ...
```

Démarrage de la réception de données avec un thread pour une execution parallèrle

6. Extract the commands:

```
def run(self):
    # ...
    while self.__running:
        line = sys.stdin.readline().rstrip() + ' '
        command = line[:line.index(' ')]
        param = line[line.index(' ')+1:].rstrip()
        if command in handlers:
            handlers[command]() if param == '' else handlers[command](param)
        else:
            print('Unknown command:', command)
```

Extraction des commandes et ses paramètres

Client/server application

1. Create a listening server:

```
s = socket.socket()
s.bind((socket.gethostname(), 6000))

s.listen()
client, addr = s.accept()
```

Renvoie un tuple avec un socket client + adresse

2. Create a client and connect (TCP):

```
s = socket.socket()

s.connect((socket.gethostname(), 6000))
```

3. Send and recv to communicate:

```
data = "Hello World!".encode()
sent = s.send(data)
if sent == len(data):
    print("Envoi complet")
```

With a loop:

```
msg = "Hello World!".encode()
totalsent = 0
while totalsent < len(msg):
    sent = s.send(msg[totalsent:])
    totalsent += sent
```

4. Receive:

```
data = s.recv(512).decode()
print('Reçu', len(data), 'octets :')
print(data)
```

Or with a loop:

```
chunks = []
finished = False
while not finished:
    data = client.recv(1024)
    chunks.append(data)
    finished = data == b''
print(b''.join(chunks).decode())
```

Serveur echo

- Boucle d'acceptation de clients

  On accepte un client à la fois

- Mise en attente des demandes tant qu'un client est traité

  Taille file d'attente modifiable avec un paramètre de `listen`

```
1  class EchoServer():
2      # ...
3
4      def run(self):
5          self.__s.listen()
6          while True:
7              client, addr = self.__s.accept()
8              print(self._receive(client).decode())
9              client.close()
10
11     # ...
```

Client echo

- Connexion au serveur echo

  Utilisation de l'adresse TCP du serveur (IP, port)

- Envoi du message texte au serveur

```
1  class EchoClient():
2      # ...
3
4      def run(self):
5          self.__s.connect(SERVERADDRESS)
6          self._send()
7          self.__s.close()
8
9      # ...
```

Communication protocols

1. Check errors:

```
s = socket.socket()
try:
    s.connect(("www.gloogle.be", 82))
except OSError:
    print('Serveur introuvable, connexion impossible.')
```

2. Types of what we want to send:

```
data = 12
print(str(data).encode())          str and int with encoe/decode
print(pickle.dumps(data))          objects with pickle (dumps/load)
print(struct.pack('I', data))      primitive data with struct (pack/unpack)
```

CherryPy

Framework CherryPy

- Définition d'une nouvelle application `WebApp`

  Lancement de l'application avec la méthode `quickstart`

- Définition d'une nouvelle route `index`
  - Décorateur `@cherrypy.expose` pour chaque route désirée
  - Construction et renvoi d'un contenu HTML

```
1  import cherrypy
2
3  class WebApp():
4      @cherrypy.expose
5      def index(self):
6          return "Hello <b>World</b>!"
7
8  cherrypy.quickstart(WebApp())
```

Go on website http://localhost:8080

1. With a parameter:

```
import cherrypy

class WebApp():
    @cherrypy.expose
    def index(self, name='World'):
        return 'Hello <b>{}</b>!'.format(name)

cherrypy.quickstart(WebApp())
```

2. Configurations:

```
import cherrypy

class WebApp():
    @cherrypy.expose
    def index(self, name='World'):
        return 'Hello <b>{}</b>!'.format(name)

cherrypy.quickstart(WebApp(), '', 'server.conf')
```

3. Default route:

```
# [...]

    @cherrypy.expose
    def default(self, attr='abc'):
        return '<h1>404</h1><p>Page not found! Fin du monde!</p>'

# [...]
```

4. To have an input:

```
# [...]

    @cherrypy.expose
    def loginform(self):
        return '''<form action="/login" method="post">
  <input type="text" name="name" />
  <input type="submit" value="Login" />
</form>'''.encode('utf-8')

# [...]
```

```
# [...]

    @cherrypy.expose
    def login(self, name):
        return 'Bonjour {} !'.format(name)

# [...]
```

5. Take a JSON:

```
# [...]

   @cherrypy.expose
   @cherrypy.tools.json_in()
   @cherrypy.tools.json_out()
   def adduser(self):
       data = cherrypy.request.json
        user = json.loads(data)
        print('> Ajout de ' + user['name'])
        return {'OK': True}

# [...]
```

Code examples

```
import socket
import sys
import threading
import string
import json
import os
import cherrypy


#Exo 1 : connect to ecam with port 80

""" Examples """
print ( socket . getfqdn ("www.google.be"))        # Fully Qualified Domain Name
print ( socket . gethostname ())                   # Nom d'hôte de la machine
print ( socket . gethostbyname ("www.google.be"))  # Hôte à partir du nom

""" connecting """
s = socket.socket()     #socket.AF_INET6, socket.SOCK_DGRAM
s.connect(('www.google.be',80))
print(s.getsockname())

""" working :"""
data = "GET / HTTP/1.0\n\n".encode()
data = b'GET / HTTP/1.0\n\n'        #other way
sent = s.send(data)
receipt = s.recv(512).decode()
print("receipt",len(receipt), "bytes :")
print(data)

""" not working :"""
s.send(b'GET / HTTP/1.0\n\n' )   #b for encoded in bytes
chunks = []
finnished = False
while not finnished:
    data = s.recv(512)
    chunks.append(data)
    finnished = data == b''
print(b''.join(chunks).decode())        #something not working here

""" closing :"""
s.close()
```

```
#Exo 2: create a chat
"""
Le serveur permet de mémoriser la liste des clients disponibles pour chatter. Il retient pour
chaque client son pseudo et son adresse IP.

– Au demarrage, Le client va se présenter au serveur, ce qui fait qu'il sera disponible pour chatter.
Il peut interroger le serveur pour obtenir la liste des clients disponibles.

– Ayant l'adresse IP d'une autre machine, une machine peut lancer un chat avec une autre en mode
peer-to-peer, tout cela indépendamment du serveur.
"""

# __running un bbooleen pour indiquer si ca tourne
# __address pour avoir l'addresse du destinataire
# __s : le socket
```

```
class Chat:
    def __init__(self, host="0.0.0.0", port=5000):
        s = socket.socket(type=socket.SOCK_DGRAM)
        s.settimeout(0.5)
        s.bind((host, port))
        self.__s = s
        print('Écoute sur {}:{}'.format(host, port))

    def run(self):
        handlers = {
            '/exit': self._exit,
            '/quit': self._quit,
            '/join': self._join,
            '/send': self._send
        }
        self.__running = True
        self.__address = None
        threading.Thread(target=self._receive).start()
        while self.__running:
            line = sys.stdin.readline().rstrip() + ' '
            # Extract the command and the param
            command = line[:line.index(' ')]
            param = line[line.index(' ')+1:].rstrip()
            # Call the command handler
            if command in handlers:
                try:
                    handlers[command]() if param == '' else handlers[command](param)
                except:
                    print("Erreur lors de l'exécution de la commande.")
            else:
                print('Command inconnue:', command)

    def _exit(self):
        self.__running = False
        self.__address = None
        self.__s.close()

    def _quit(self):
        self.__address = None

    def _join(self, param):
        tokens = param.split(' ')
        if len(tokens) == 2:
            try:
                self.__address = (tokens[0], int(tokens[1]))
                print('Connecté à {}:{}'.format(*self.__address))
            except OSError:
                print("Erreur lors de l'envoi du message.")

    def _send(self, param):
        if self.__address is not None:
            try:
                message = param.encode()
                totalsent = 0
                while totalsent < len(message):
                    sent = self.__s.sendto(message[totalsent:], self.__address)
                    totalsent += sent
            except OSError:
                print('Erreur lors de la réception du message.')

    def _receive(self):
        while self.__running:
            try:
                data, address = self.__s.recvfrom(1024)
                print("[{}] {}".format(address, data.decode()))
            except socket.timeout:
                pass
            except OSError:
                return
```

```
if __name__ == '__main__':
    if len(sys.argv) == 3:
        Chat(sys.argv[1], int(sys.argv[2])).run()
    elif len(sys.argv) == 2:
        Chat(port=int(sys.argv[1])).run()
    else:
        Chat().run()
```

Exo 2 not finnished!!!

```python
#Exo 3 : cherrypy
import webbrowser


print(socket.gethostbyname(socket.gethostname()))
class WebApp():
    @cherrypy.expose
    def index(self):
        return '<b>YYOOOOOOOOOOOOOOOOOOOOO<b>!'

site = "http://localhost:8080"
webbrowser.open(site)
print('browser started !')


cherrypy.server.socket_host = "0.0.0.0"
cherrypy.quickstart(WebApp())


""" Pour se connecter depuis un autre pc : ipadress:8080"""
```

Chapter 7&8: algorithms and AI

Algorithm

- The point of it is to solve a problem in the form of a function

Algorithm 1: Recherche si un élément se trouve dans une liste

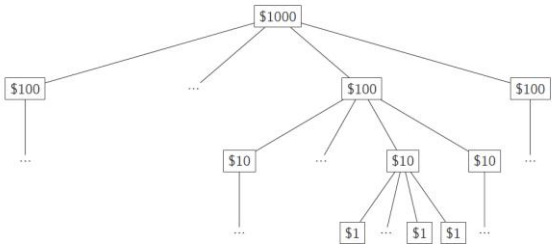
```
if n = 0 then
    return false
foreach e ∈ L do
    if e = elem then
        return true
return false
```

pseudo code

- Trying to optimize the algorithm

Recursion

- Stratégie de diviser pour régner (*divide-and-conquer*)
  - *Décomposition du problème original en sous-problèmes*
- Cas de base simple et cas récursif à décomposer
  - *Représentation graphique avec un arbre de solution*



- Processus de récursion découpe en sous-problèmes
  - Dont la structure est la même que le problème original
  - Plus simples à résoudre
- Décompositions successives du problème original
  - *Jusqu'à avoir un sous-problème qui se résout directement*

- Problème candidat à une solution récursive

1. On peut **décomposer** le problème original en *instances plus simples* du même problème
2. Les sous-problèmes doivent finir par *devenir suffisamment simples* que pour être **résolus directement**
3. On peut **combiner** les solutions des sous-problèmes pour *produire la solution* du problème original

Cas récursif $n! = n \cdot (n-1)!$

Factorielle de n se calcule à partir de celle de $n-1$

```
def fact(n):
    if n == 0:
        return 1
    return n * fact(n - 1)
```

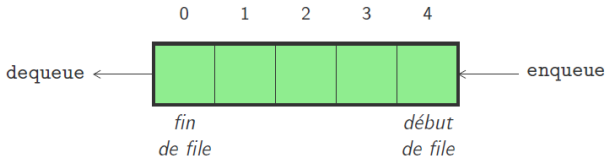
Parents and children

1. FIFO (first in first out):

   Le premier élément qui a été ajouté sera le premier à sortir

   - Opérations possibles

`size`	donne la taille de la file
`isEmpty`	teste si la file est vide
`front`	récupère l'élément en début de file
`enqueue`	ajoute un élément en fin de file
`dequeue`	retire l'élément en début de file

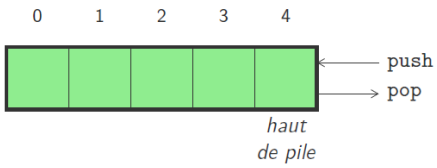


2. LIFO (last in first out):

   Le dernier élément qui a été ajouté sera le premier à sortir

   - Opérations possibles

`size`	donne la taille de la pile
`isEmpty`	teste si la pile est vide
`top`	récupère l'élément en haut de la pile
`push`	ajoute un élément en haut de la pile
`pop`	retire l'élément en haut de la pile



3. Tree:

- Eléments d'un arbre organisés de manière hiérarchique

  Un arbre est un ensemble de nœuds (qui contiennent les valeurs)

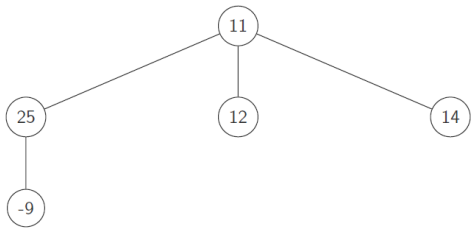
- Chaque nœud possède un parent et zéro ou plusieurs enfants

  Sauf la racine de l'arbre qui n'a pas de parent

- Opérations possibles

`size`	donne la taille de l'arbre
`value`	récupère la valeur stockée à la racine de l'arbre
`children`	récupère la liste des sous-arbres enfants de la racine
`addChild`	ajoute un sous-arbre comme enfant à la racine

```python
import copy

class Tree:
    def __init__(self, value, children=[]):
        self.__value = value
        self.__children = copy.deepcopy(children)

    @property
    def value(self):
        return self.__value

    @property
    def children(self):
        return copy.deepcopy(self.__children)

    def addChild(self, tree):
        self.__children.append(tree)

    # ...
```




```python
t1 = Tree(-9)
t2 = Tree(25, [t1])
t3 = Tree(12)
t4 = Tree(14)

t = Tree(11, [t2, t3, t4])
```

Find the size:

```python
    @property
    def size(self):
        result = 1
        for child in self.__children:
            result += child.size
        return result
```

4. Backtracking:
   - Faire une tentative de séquences de choix
     - Possibilité de faire marche arrière par rapport à un choix
     - Exploration de nouvelles décisions
   - La récursion permet de faire facilement du backtracking
5. Lookahead:
   - Explorer un maximum de coups possibles à l'avance

     Sélectionner le coup qui mène à la meilleure situation
   - Pas toujours possible d'explorer tous les coups

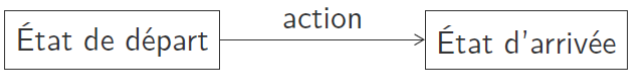
     Trouver le moins pire étant donné une contrainte temporelle
   - Deux notions clés
     - L'**état du jeu** représente la situation de ses joueurs
     - Un **coup** fait la transition entre deux états

Research problem

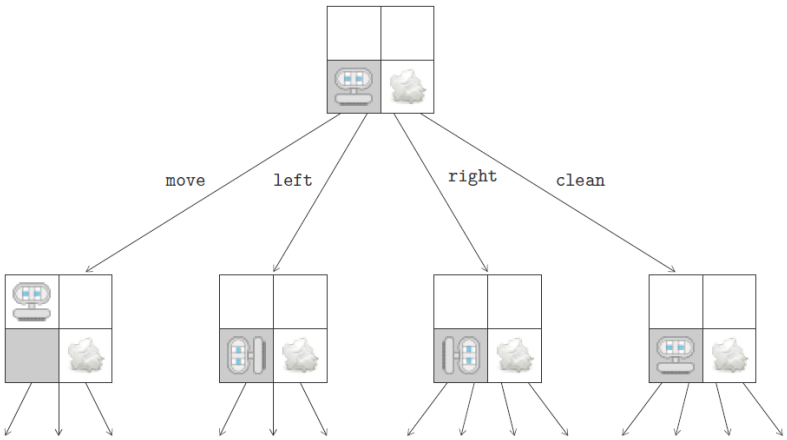
1. First, the system is in an initial state
2. Action:
   - Une action est effectuée sur l'environnement

     Modification de l'état de l'environnement suite à l'action
   - Ensemble d'actions possibles pour chaque état

     Des actions peuvent être indisponibles dans certains états



3. Every possible action (all children):



4. Check the best way

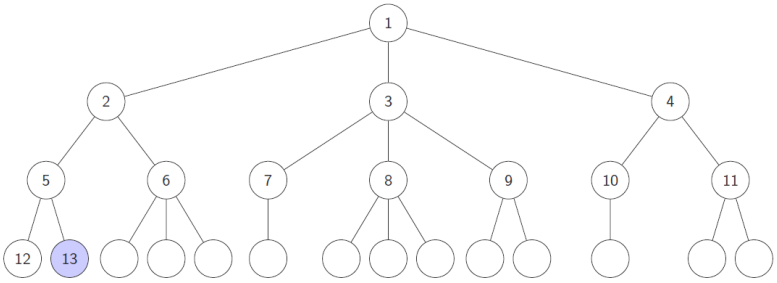
Search algorithms

1. Breadth-first Search:
   - Exploration successives des successeurs en largeur

     D'abord les successeurs d'un nœuds avant leurs successeurs...

   - Exploration de l'arbre d'exécution par niveaux

     Algorithme complet, mais pas forcément optimal



2. Uniform-cost Search:
   - Exploration via l'action qui a le cout le plus faible

     Permet d'explorer d'abord le chemin de cout total minimal

   - Complétude et optimalité assurées si $c(q, a) > \varepsilon$ pour $\varepsilon > 0$

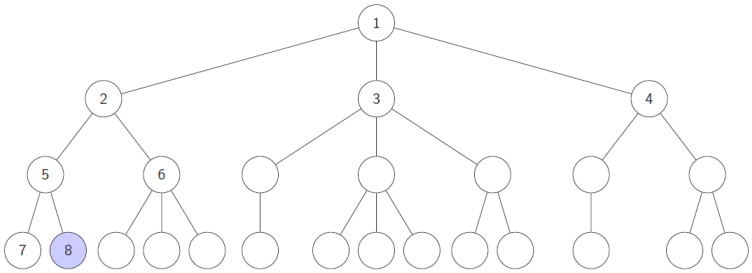
     Nœuds parcourus en ordre croissant du cout du chemin associé

3. Depth-first Search:
   - Exploration d'abord en profondeur

     D'abord explorer le nœud non exploré le plus profond

   - Descente jusqu'à une feuille de l'arbre

     Pas complet (peut être coincé dans une boucle), ni optimal



4. Depth-limited Search:
   - Ajout d'une profondeur maximale d'exploration ℓ
     - Pas complet si $\ell < d$ (solution hors de portée)
     - Pas optimal si $\ell > d$ (peut rater la solution la moins profonde)
   - Complexité temporelle en $\mathcal{O}(b^\ell)$ et spatiale en $\mathcal{O}(b\ell)$

     Cas particulier de Depth-First Search avec $\ell = \infty$

   - Deux types d'échecs différents
     - **Failure** lorsque pas de solutions
     - **Cutoff** lorsque pas de solutions dans la limite ℓ

5. Iterative deepening depth-first search:
   - Augmentation progressive de la profondeur maximale

     Depth-Limited Search avec successivement $\ell = 0$, $\ell = 1$...

   - Solution optimale trouvée lorsque $\ell = d$

     Algorithme complet et optimal

6. Bidirectional Search:
   - Effectuer deux recherches en parallèle

     Une depuis l'état initial et une depuis le(s) objectif(s)

   - Nécessite que la fonction prédécesseur soit disponible

     Facile si les actions sont réversibles

7. Greedy best-first Search:
   - Choix du nœud le plus proche de l'objectif

     En utilisant $f(n) = h(n)$

   - L'heuristique est choisie en fonction du problème

     Souvent une mesure de distance vers l'objectif

8. A* Search:
   - A^* (prononcé « *A-star* ») combine deux fonctions
     - $g(n)$ donne le cout d'avoir atteint n
     - $h(n)$ heuristique du cout pour atteindre l'objectif depuis n
   - Fonction d'évaluation $f(n) = g(n) + h(n)$

     Cout estimé pour atteindre l'objectif en passant par n

9. Adversarial search:
   - Recherche de solution pour des jeux avec deux adversaires

     Deux joueurs appelés `MIN` et `MAX` (joue en premier)

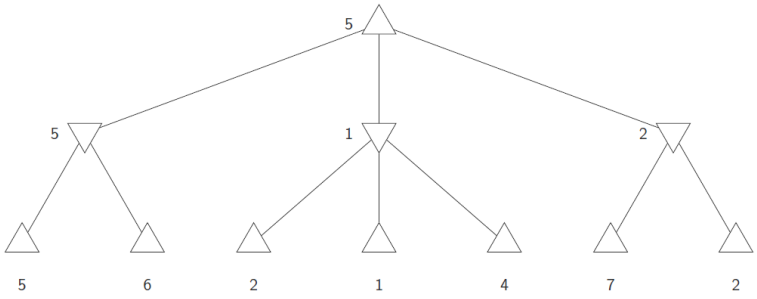
   - Définition comme un problème de recherche
     - **État initial** : position sur le plateau et joueur qui commence
     - **Fonction successeur** : liste de paires $(move, state)$
     - **Test terminal** : teste si le jeu est terminé (état terminaux)
     - **Fonction d'utilité** : donne une valeur aux états terminaux
   - Jouer le meilleur coup possible à chaque tour

     En supposant que le joueur en face suit une stratégie parfaite

Minimax

- Arbre du jeu avec $\triangle$ pour `MAX` et $\triangledown$ pour `MIN`

  `MAX` choisit toujours le coup qui maximise la valeur minimax



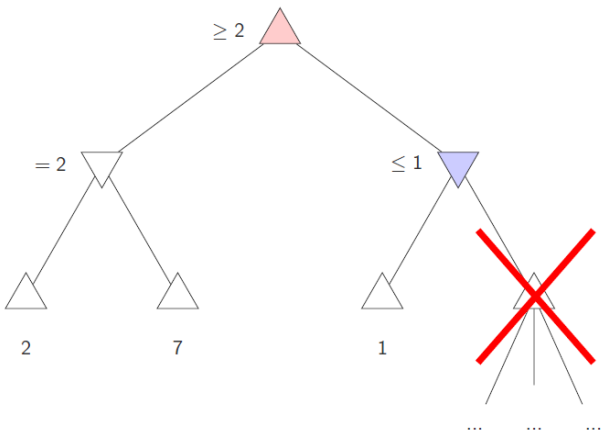
Alpha-beta pruning

- Éviter d'explorer un sous-arbre lorsque ce n'est pas nécessaire

  On ne sait pas faire mieux que la valeur minimax actuelle

- Deux situations de simplification possibles
  - α plus grande borne inférieure pour `MAX`
  - β plus petite borne supérieure pour `MIN`

- Mémorisation des bornes durant l'exploration de l'arbre



Examples:

```
def minimax(state,depth,alpha,beta,player):
    """
    param: state, the board, that changes further when we iterate it in the loop
    param: depth, it's the depth we chose outside the function and will decrease for each iteration we do
        the starting depth is for ex 4, it's the root, and 1 is the leaves. It can't go under 1
    param: alpha and beta for pruning useless branches and optimizing
    param: player stands for human or computer turn, as it will maximize at comp turn and minimize at human turn
    return : returns the best move of its children
    """

    temp_board = msgpack.packb(state)
    base_eval = eval(state)     #basic evaluation of the board

    if player == computer:
        best = {"from":list,"to":list,"tscore":-infinity}       # - inf because we want the algo to maximize for the computer
    else :
        best = {"from":list,"to":list,"tscore":+infinity}

    if depth == 0 or game_over(state):          #when we're at the level under the leaves, leaves depth = 1
        return best

    for move in all_moves(state):      #the loop that iterates itself when going inside it's children
        x, y = move["from"]
        a,b = move["to"]

        new_state = msgpack.unpackb(temp_board)
        set_move(x,y,a,b,new_state)     #creating a copy and setting a new move in the copy

        if depth == 1: # if leaf
            move["tscore"] = base_eval + diff(x, y, a, b, new_state)    #checks the base evaluation from parent and adding the difference with the child

        else:       #we iterate the function minimax by taking the best score the children returned
            move["tscore"] = minimax(new_state,depth-1,alpha,beta,-player)["tscore"] # = the best score from children


        # compare score with adjacent nodes
        if player == computer:  #maximize

            if best["tscore"]< move["tscore"]:
                best = move     #max value
            alpha = max(alpha,best["tscore"])
        else :
            if best["tscore"]> move["tscore"]:
                best = move     #min value
            beta = min(beta,best["tscore"])

        if beta <= alpha:
            break

    return best #returns only the best score of the children
```

Chapter 9&10: Multicore and multithreading

Multiprocessing:

- Un **processus** est un programme en cours d'exécution

  Plusieurs processus peuvent exister pour un même programme

- Plusieurs caractéristiques liés aux processus
  - Processus gérés par le système d'exploitation
  - Trois flux connectés : entrée/sortie/erreur standard
  - Code de retour après exécution (0 terminaison normale)
- **Informations** sur le processus avec le module `sys`

Lancer un processus

- **Processus** représenté par un objet Process

  Code du processus défini par une fonction passée au constructeur

- **Méthode start** pour lancer le processus

```
import multiprocessing as mp

def sayhello (name):
    print ('Hello', name)
```

```
if __name__ == '__main__':
    proc = mp.Process(target = sayhello, args =('Dan',))
    proc.start()

    proc.join() # Attendre la fin du processus
    print('Terminé avec code', proc.exitcode)
```

A utiliser absolument

d'office dans un tuple

Communication interprocessus

- Possibilité d'échanger des objets entre processus

  Permet une communication bidirectionnelle entre deux processus

- Deux constructions différentes proposées
  - File de communication `(Queue)`
  - Tube de communication `(Pipe)`
- Attendre la fin d'un processus avec la méthode `join`

  Méthode `join` bloquante

Communication par `Queue`

- Création de la `Queue` et passation au processus fils

  Méthodes `put` et `get` pour écrire et lire dans la file

```
def compute(q):
    q.put('Hey !')

if __name__ == '__main__':
    q = mp.Queue()
    proc = mp.Process(target = compute, args =(q,))
    proc.start()
    print(q.get())

    proc.join() # Attendre la fin du processus
    print('Terminé avec code', proc.exitcode)
```

va etre bloquant tant que l'autre processus donne pas une value

Communication par `Pipe`

pour communiquer dans les deux sens

- Création du `Pipe` et passation d'un bout au processus fils

  Méthodes `send` et `recv` pour envoyer et recevoir des données

```
import multiprocessing as mp

def compute(child):
    child.send('Hey !')
    child.close()

if __name__ == '__main__':
    parent, child = mp.Pipe()
    proc = mp.Process(target = compute, args =(child,))
    proc.start()
    print(prent.recv())

    proc.join() # Attendre la fin du processus
    print('Terminé avec code', proc.exitcode)
```

Pool de workers

- Parallélisme par exploitation des multi-processeurs

  Permet de lancer plusieurs processus localement ou à distance

- Parallélisme de données à l'aide d'objets Pool

```
import multiprocessing as mp


def compute(data):
    return data ** 2


if __name__ == '__main__':
    with mp.Pool(3) as pool:
        print(pool.map(compute, [1, 7, 8, -2]))
```

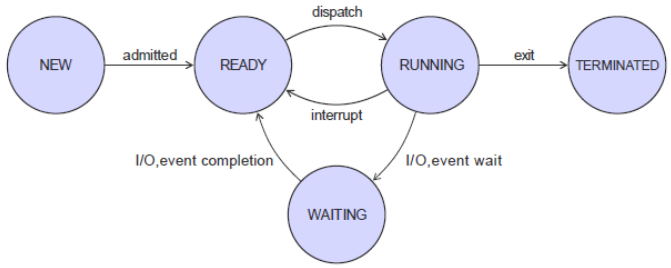
3 pour 3 coeurs

Multithreading:

- Processus léger à l'intérieur d'un processus

  Plusieurs états lors de l'exécution d'un thread

- Un thread terminé ne peut pas être redémarré



Planifier une action

- Planification d'une exécution à l'aide d'un objet `Timer`

  Choix de la fonction à exécuter et du temps d'attente

- Création d'un objet Timer puis appel à la méthode `start`

```
import threading as th

def sayhello(name) :
    print('Hello', name)

t = th.Timer(5.5, sayhello, args = ('Bob',))
t.start()
print('Timer started')
```

timer pas bloquant

Lancer un Thread

- Thread représenté par un objet `Thread`

  Code du thread défini par une fonction passée au constructeur

- Méthode `Méthode` pour lancer le thread

```
import threading as th

def sayhello(name):
    print('Hello', name)

thread = th.Thread(target = sayhello, args = ('Tom',))
thread.start()

thread.join() # Attendre la fin du thread
print('Thread', thread.name, 'terminé')
```

Pool d'executor

- Parallélisme par création de processus légers

  Création plus rapide par rapport à un processus

- Un thread possède un nom

  Accessible par l'attribut `name`

```
import concurrent.futures as cf

def compute(data):
    return data ** 2

with cf.ThreadPoolExecutor(3) as executor :
    print(list(executor.map(compute, [1, 7, 8, -2])))
```

3 threads