Python 3.x

Structures

Une classe

```
class Dog(Animal):
  def __init__(self):
    # ...

  def hello(self):
    # ...
```

Une fonction

```
def ma_fonction(arg1, arg2):
  return 3
```

Les conditions

```
if a == b:
  # faire quelque chose
elif b == c:
  # autre chose
else:
  # encore autre chose
```

Les conditions ternaires

```
a = 5 if c == b else 8
# Si c == b, a = 5 sinon 8
```

Gestion des exceptions

```
try:
  # quelque chose
except:
  # autre chose
```

Conversions

Conversion entiers / chaines de caractères

```
int("1853") * 2 # 3706
str(1853) * 2   # "18531853"
```

Initialisations

Initialiser une liste 6 éléments à 3

```
l = [3,]*6
# [3, 3, 3, 3, 3, 3]
```

Initialiser 4 variables à None

```
a,b,c,d = (None,)*4
# a=None, b=None, c=None, d=None
```

Générer une liste de carrés

```
[i**2 for i in range(1,6)]
# Genere [1, 4, 9, 16, 25]
```

Générer un tableau 2D (une matrice) de 2 par 3 à -1

```
[[-1 for x in range(2)] for y in range(3)]
# [[-1, -1],
#  [-1, -1],
#  [-1, -1]]
```

Iterables

Tout les exemples sont présentés avec une chaine de caractère mais fonctionnent également avec une liste ou d'autres iterables.

Itérer sur les charactères d'une liste

```
for carac in "hello world":
  print(carac, end="-")
# h-e-l-l-o- -w-o-r-l-d-
```

Accéder à des caractères d'une liste

```
"hello world"[2]  # 3eme element "l"
"hello world"[-1] # dernier element "d"
```

Accéder à des sous chaines de caractère

```
chaine = "hello world"
chaine[1:5]   # "ello"
chaine[-5:-1] # "worl"
chaine[-5:]   # "world"
chaine[4:]    # "o world"
```

Inverse la chaîne de caractère

```
chaine[::-1] # "dlrow olleh"
```

Listes

Ajout et concaténation

```
[1, 2, 3].append(4)   # [1, 2, 3, 4]
[1, 2] + [3, 4]       # [1, 2, 3, 4]

a = [1, 2]
a += [3, 4]
a.extend([5, 6])
a.append(7)
# a = [1, 2, 3, 4, 5, 6, 7]
```

Associer plusieurs listes

```
zip([1, 2, 3], [4, 5, 6])
# [(1, 4), (2, 5), (3, 6)]
```

Récupérer l'index d'un élément

```
["foo", "bar", "baz"].index("bar")
# 1
```

Chaines de caractères

Conversion code ASCII / caractère

```
chr(97)  # ’a’
ord(’a’) # 97
```

Dictionnaire

Vérifier l'existence d'une clé

```
dic = {"a": 1}
if "a" in dic:
  # Verifie si la clef a existe
  pass
```

Itérer sur un dictionnaire

```
dic = {"a": 1, "b": 2}
for cle, valeur in dic.items():
  print(cle, valeur)
```

Tuple

Tuple packing et unpacking

```
t = 12345, 54321, ’hello!’
x, y, z = t
```

Cas particuliers (Tuple de 0 et 1 élément)

```
empty = ()
singleton = ’hello’,
# notez la derniere virgule
```

Set

Un set ne contient qu'une seule fois chaque valeur et n'est pas ordonné.

```
{8, 9, 9, 1}
# {9, 8, 1}
```

Entrées / Sorties

Pour lire une ligne sur l'entrée standard :

```
input() # stdin
```

Pour lire sur l'entrée standard jusqu'à un EOF (End Of File) :

```
import sys

for line in sys.stdin:
  print(line)
```

Vous pourrez alors executer votre application avec "python3 monapp.py ¡ fichier.txt" ou "python3 monapp.py" et écrire ce que vous voulez puis terminer par un CTRL + D

Pour écrire sur la sortie standard :

```
print(x, y, z) # print sur stdout
print("fatal error", file=sys.stderr)
# print sur stderr
```

Fonctionnel

Réduction (reduce)

```
from functools import reduce
reduce(lambda x, y: x*y, [2, 3, 4])
# 2 * 3 * 4 = 24
```

Filtre (filter)

```
list(filter(lambda x: x > 2, [1,2,3,4]))
# [3, 4]

[n for n in [1, 2, 3, 4] if n > 2]
# [3, 4]
```

Association (map)

```
list(map(lambda x: x**2, [2, 3, 4]))
# [4, 9, 16]

[n**2 for n in [2, 3, 4]]
# [4, 9, 16]
```

Mathématiques

Récupérer le minimum ou le maximum de plusieurs valeurs.

```
min(3, 5)       # 3
min(3, 2, 8, 7) # 2
min([13, 5, 8]) # 5
max(6, 3)       # 3
...
```

A la puissance n

```
i, n = (3, 2)
i ** n     # 9
pow(i, n) # 9
```

Valeur absolue

```
abs(-5) # 5
```

Tri

Retourner une nouvel iterable trié (Fonctionne avec tout iterable)

```
sorted([9,12,2])
# [2, 9, 12]

sorted({"F": 0, "D": 0, "A": 0, "B": 0})
# ['A', 'B', 'D', 'F']

sorted([9,12,2], reverse=True)
# [12, 9, 2]
```

Trier une liste (seulement)

```
a = [5, 2, 8]
a.sort()
# a = [2, 5, 8]
```

Threads et Queue

```
from Queue import Queue
from threading import Thread

def listener(q):
  while True:
    print(q.get())

q = Queue()
t = Thread(target=listener, args=(q))

t.start()
q.put("hello")
```

Réseau

```
import socket, select

sock = socket.socket( \
  socket.AF_INET, \
  socket.SOCK_STREAM)
rlist = []

sock.bind(('0.0.0.0', 1025))
sock.listen()
while True:
  rd, wr, err = select.select(rlist, [], [])
  for s in rd:
    if s is sock:
      client_socket, address = sock.accept()
      rlist.append(client_socket)
    else:
      data = s.recv(1024)
      if data: print(data); sock.send("OK")
      else: s.close(); rlist.remove(s)
```

HTTP

http://flask.pocoo.org/docs/0.11/quickstart/
Créer un dossier /static pour servir des fichiers.
Créer un dossier /templates pour mettre les templates au format JINJA2.

```
<h1>{{ name }} </h1>
```

Code d'exemple avec Flask

```
from flask import Flask, request, \
        render_template, url_for, session
app = Flask(__name__)

@app.route("/user")
@app.route("/user/<username>", \
        methods=['GET', 'POST'])
def hello(username=None):
    if request.method == 'POST':
        # request.form['hello']
        # session['username'] = xx
        return render_template( \
        'hello.html', name=username)
    else:
        return "Hello "+username+" !"

app.run()
```

Stocker données

TODO

Hash et encodage

base64
md5
hash

Programmation dynamique

Deux méthodes "systématiques" :
1. Librairie standard

```
from functools import lru_cache
...
# max_size le nombre d'elements max
# du cache ou None (cache "infini")
@lru_cache(max_size=42)
def anything(*args):
....
```

2. homemade

```
from collections import defaultdict
def dynamic(f):
    cache = defaultdict(lambda:-1)
    def is_known(*args):
        if cache[args] == -1:
            cache[args] = f(*args)
        return cache[args]
    return is_known

@dynamic
def anything(*args):
    ....
```

La bissection - dichotomie

```
def bisect(func, low, high, desired, iter):
    for i in range(iter):
        midpoint = (high - low) / 2.0 + low
        if func(midpoint) > desired:
            high = midpoint
        else:
            low = midpoint

    return midpoint
```

Union Find

```
def MakeSet(x):
     x.parent = x
     x.rank   = 0

def Union(x, y):
     xRoot = Find(x)
     yRoot = Find(y)
     if xRoot.rank > yRoot.rank:
         yRoot.parent = xRoot
     elif xRoot.rank < yRoot.rank:
         xRoot.parent = yRoot
     elif xRoot != yRoot:
         yRoot.parent = xRoot
         xRoot.rank = xRoot.rank + 1

def Find(x):
     if x.parent == x:
        return x
     else:
        x.parent = Find(x.parent)
        return x.parent

class Node:
    def __init__ (self, label):
        self.label = label
    def __str__(self):
        return self.label

l = [Node(ch) for ch in "abcdefg"]
[MakeSet(node) for node in l]
```

```
Union(l[0],l[2])
sets = [str(Find(x)) for x in l]
```

Segment Tree

```
def buildTree(root, Tree, start, end, inp):
    if start == end:
        Tree[root] = input[start]
        return Tree[root]

    mid = start + (end - start) / 2
    leftMin = buildTree(
      root * 2 + 1, Tree, start, mid, inp)
    rightMin = buildTree(
      root * 2 + 2, Tree, mid + 1, end, inp)
    Tree[root] = min(leftMin, rightMin)
    return Tree[root]

def rangeQUtil(root, Tree, start, end, qs, qe):
    if qe < start or qs > end:
        return float("inf")

    if qs <= start and qe >= end:
        return Tree[root]

    int mid = start + (end - start) / 2
    int leftMin = rangeQUtil(
      root * 2 + 1, Tree, start, mid, qs, qe)
    int rightMin = rangeQUtil(
      root * 2 + 2, Tree, mid + 1, end, qs, qe)
    return min(leftMin, rightMin)

# Initialize
inp = [0,1,2,3,4,5,6,7,8,9]
tr = [-1 for x in range(len(input)*2+1)]
tr = buildTree(0, tr, 0, len(input)-1, inp)

# Query
n = len(inp) - 1; qstart = 0; qend = 5
rangeQUtil(0, tr, 0, n, qstart, qend)
```

Dijkstra

```
import heapq

class PriorityQueue:
  def __init__(self):
    self.elements = []
  def empty(self):
    return len(self.elements) == 0
  def put(self, item, priority):
    heapq.heappush(self.elements, (priority, item))
  def get(self):
    return heapq.heappop(self.elements)[1]

class SimpleGraph:
  def __init__(self):
    self.edges = {}
    self.weights = {}
  def neighbors(self, id):
    return self.edges[id]
  def cost(self, from_node, to_node):
    return self.weights[(from_node, to_node)]

def dijkstra_search(graph, start, goal):
  frontier = PriorityQueue()
  frontier.put(start, 0)
  came_from = {}
  cost_so_far = {}
  came_from[start] = None
  cost_so_far[start] = 0

  while not frontier.empty():
    current = frontier.get()

    if current == goal:
      break

    for next in graph.neighbors(current):
      new_cost = cost_so_far[current] \
        + graph.cost(current, next)

      if next not in cost_so_far or \
        new_cost < cost_so_far[next]:

        cost_so_far[next] = new_cost
        priority = new_cost
        frontier.put(next, priority)
        came_from[next] = current

  return came_from, cost_so_far

def reconstruct_path(came_from, start, goal):
  current = goal
  path = [current]
  while current != start:
    current = came_from[current]
    path.append(current)
  path.append(start)
  path.reverse()
  return path

example_graph = SimpleGraph()
example_graph.edges = {
    'A': ['B'],
    'B': ['A', 'C', 'D'],
    'C': ['A'],
    'D': ['E', 'A'],
    'E': ['B']
}
example_graph.weights = {
    ('A', 'B'): 5,
    ('B', 'A'): 4,
    ('B', 'C'): 6,
    # ...
}

from, cost = dijkstra_search(example_graph, 'A', 'E')
reconstruct_path(from, 'A', 'E')
```

Breath-First Search

Graphe de référence

```
graph = {'A': set(['B', 'C']),
         'B': set(['A', 'D', 'E']),
         'C': set(['A', 'F']),
         'D': set(['B']),
         'E': set(['B', 'F']),
         'F': set(['C', 'E'])}
```

Composantes connexes (tous les points connectés à ce noeud)

```
def bfs(graph, start):
  visited, queue = set(), [start]
  while queue:
    vertex = queue.pop(0)
    if vertex not in visited:
      visited.add(vertex)
      queue.extend(graph[vertex] - visited)
  return visited

bfs(graph, 'A')
# {'B', 'C', 'A', 'F', 'D', 'E'}
```

Recherche de chemins

```
def bfs_paths(graph, start, goal):
  queue = [(start, [start])]
  while queue:
    (vertex, path) = queue.pop(0)
    for next in graph[vertex] - set(path):
      if next == goal:
        yield path + [next]
      else:
        queue.append((next, path + [next]))

list(bfs_paths(graph, 'A', 'F'))
# [['A', 'C', 'F'], ['A', 'B', 'E', 'F']]
```

Tableaux ASCII

Lettres minuscules

dec	char	dec	char	dec	char
97	a	106	j	115	s
98	b	107	k	116	t
99	c	108	l	117	u
100	d	109	m	118	v
101	e	110	n	119	w
102	f	111	o	120	x
103	g	112	p	121	y
104	h	113	q	122	z
105	i	114	r		

Lettres majuscules

dec	char	dec	char	dec	char
65	A	74	J	83	S
66	B	75	K	84	T
67	C	76	L	85	U
68	D	77	M	86	V
69	E	78	N	87	W
70	F	79	O	88	X
71	G	80	P	89	Y
72	H	81	Q	90	Z
73	I	82	R		

Algorithmes

ROT N

Génération de nombres premiers

Dynamic erathostene

```python
primes = [2, 3]

def bumblebee(n):
    prime = True
    i = primes[-1]
    while primes[-1] < n:
        prime = True
        for p in primes:
            if i % p == 0:
                prime = False
                break
        if prime:
            primes.append(i)
        i += 2
```