

Initiation au python

Groupe Montpellier Bio-Stat



Montpellier
Bio-Stat



<https://groupes.renater.fr/wiki/montpellier-biostat>



MUSE

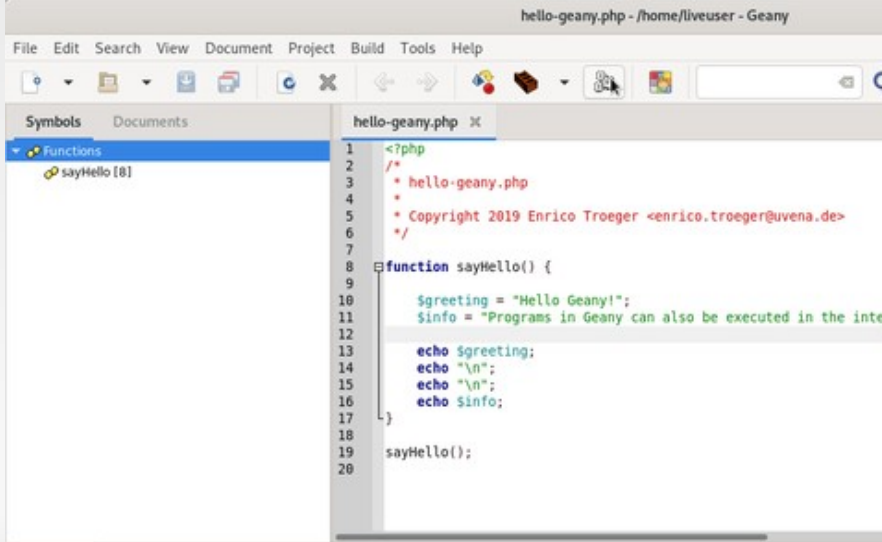


MONTPELLIER UNIVERSITÉ D'EXCELLENCE



Ressources

- Cours python de Pierre Giraud (Débutant)
 - ➔ <https://www.pierre-giraud.com/python-apprendre-programmer-cours/>
- Cours python de Nicolas Barrier (Avancé)
 - ➔ <https://github.com/umr-marbec/python-training>
- IDE (Integrated Development Environment)
 - ➔ Geany
 - ➔ Eclipse, Spyder, Jupyter Notebook
 - ➔ N'importe quel bloc-note



The screenshot shows the Geany IDE interface. The main editor window displays a PHP file named 'hello-geany.php' with the following code:

```
1 <?php
2 /*
3  * hello-geany.php
4  * Copyright 2019 Enrico Troeger <enrico.troeger@uvena.de>
5  */
6
7
8 function sayHello() {
9
10     $greeting = "Hello Geany!";
11     $info = "Programs in Geany can also be executed in the inte
12
13     echo $greeting;
14     echo "\n";
15     echo "\n";
16     echo $info;
17 }
18
19 sayHello();
20
```

The left sidebar shows the 'Symbols' pane with a 'Functions' section containing 'sayHello (8)'. The bottom pane shows the terminal output:

```
[liveuser@localhost-live ~]$
[liveuser@localhost-live ~]$ /bin/sh /tmp/geany_run_script_QKJB3Z.sh
Hello Geany!

Compiler Programs in Geany can also be executed in the integrated terminal, as seen below

Messages -----
(program exited with code: 0)
[liveuser@localhost-live ~]$
```

Citations (véridiques) d'informaticiens

Citations (véridiques) d'informaticiens

- « Un bon informaticien est une feignasse »

Citations (véridiques) d'informaticiens

- « Un bon informaticien est une feignasse »
- « Si ça fait coin-coin, c'est un canard »

Citations (véridiques) d'informaticiens

- « Un bon informaticien est une feignasse »
- « Si ça fait coin-coin, c'est un canard »
- « Il est impossible de faire fonctionner un code python du premier coup »

Les Bases

Les Bases Helloworld !

```
# coding: utf-8
"""Python3.6"""

#####
'''
V1-2020/11/23
Print "Helloworld"

'''
#####
#MAIN
print("Helloworld!")
```

Les Bases

Helloworld !

```
#!/usr/bin/env python3
"""Python3.6"""

#####

'''
V1-2020/11/23
Print "HelloWorld"
'''

#####

#MAIN
print("HelloWorld")
```

- Commentaires : partie du code qui ne sera pas exécutée
 - ➔ #, commente tout ce qui se situe après. Monoligne
 - ➔ ''' , commente tout ce qui se situe au milieu. Multiligne
 - ➔ """ , commente tout ce qui se situe au milieu. Multiligne

Les commentaires permettent de détailler le fonctionnement du code, pour le comprendre rapidement même des années plus tard

Les Bases Helloworld !

```
# coding: utf-8
```

```
"""y h n 6"""
```

```
#####  
'''
```

```
V1-2020/11/23
```

```
Print "Helloworld"
```

```
'''
```

```
#####  
#MAIN
```

```
print("Helloworld!")
```

- Encodage : permet de signaler l'encodage des caractères

L'utf-8 est l'encodage par défaut des scripts pythons 3+, mais pas des scripts python 2.7-

L'utf-8 : **Universal** Character Set Transformation Format.

Les Bases Helloworld !

```
# coding: utf-8
"""Python3.6"""


#####
'''
V1-2020/11/23
Print "Helloworld"

'''
#####
#MAIN
print("Helloworld!")
```

- Mot-clef : expression réservée car utilisée par le langage python
- Fonction : Mot-clef(liste d'argument), effectue une action pré-programmées

Python possède un corpus prédéfinie de fonctions, auquel s'ajoute les fonctions des packages et celles définies par l'utilisateur

Les Bases Helloworld !



```
# coding: utf-8
"""Python3.6"""

#####
'''
V1-2020/11/23
Print "Helloworld"

'''
#####
#MAIN
print("Helloworld!")
print("Byebyeworld!")
```

- Un script s'exécute et effectuant les tâches listées ligne par ligne.

Les Bases Helloworld !

```
# coding: utf-8  
"""Python3.6"""
```

```
#####
```

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

```
#####  
#MAIN  
test  
print("Helloworld!")  
print("Byebyeworld!")
```

- Lorsqu'une ligne provoque une erreur, le script s'interrompt immédiatement

Toujours lire le message d'erreur !

Les Bases

Les variables

Les Bases

Les variables

```
# coding: utf-8
"""Python3.6"""

#####
'''
V1-2020/11/23
Print "Helloworld"

'''
#####
#MAIN
test="Helloworld!"
print(test)
```

- Une variable est un objet avec un nom unique, auquel on peut se référer via ce nom plus loin dans le code

Les Bases

Les variables

```
# coding: utf-8
"""Python3.6"""

#####
'''
V1-2020/11/23
Print "Hello world"

'''
#####
#MAIN
test="Hello world!"
test="Byebyeworld!"
print(test)
```

- Si un nom de variable est réutilisé, le nouveau contenu écrase le précédent contenu

Ne jamais réutiliser un nom de variable si ce n'est pas nécessaire (ou si vous ne savez pas **parfaitement** ce que vous faites)

Les Bases

Les variables – variables simples

- Les chaînes de caractères : string

```
test="Hello world!"
```

- Les chiffres entiers : int

```
test=42
```

- Les chiffres réels : float

```
test=3.14
```

- Les booléens : bool

```
test=True
```

Python ne force pas le type d'une variable. Une variable peut toujours être redéfini en un autre type

Les Bases

Les variables – variables simples

- Les chaînes de caractères : string

```
sTest="Helloworld!"
```

- Les chiffres entiers : int

```
iTest=42
```

- Les chiffres réels : float

```
fTest=3.14
```

- Les booléens : bool

```
bTest=True
```

Un nom de variable doit être composé uniquement de caractères alphanumériques, à l'exception de l'underscore « _ »

Les Bases

Les variables – assignation multiples

```
iValeurA=iValeurB=4
print(iValeurA) # => 4
print(iValeurB) # => 4

iValeurA, iValeurB=3, 8
print(iValeurA) # => 3
print(iValeurB) # => 8

iValeurA=1 ; iValeurB=9
print(iValeurA) # => 1
print(iValeurB) # => 9
```

Les assignations multiples sont moins lisibles et peuvent compliquer la compréhension du code et la recherche de bug

Pour l'interpréteur python, le symbole point-virgule « ; » équivaut à un retour à la ligne

Les Bases

Les variables – les opérations

```
iValeurA=4
iValeurB=2
print (iValeurA+iValeurB) # => 6
print (iValeurA-iValeurB) # => 2
print (iValeurA*iValeurB) # => 8
print (iValeurA/iValeurB) # => 2
iValeurC=3
print (iValeurA%iValeurC) # => 1
print (iValeurA**iValeurC) # => 64
```

```
sMotA="Hello"
sMotB="world!"
print (sMotA+sMotB) # => 'HelloWorld!'
iValeurA=4
print (sMotA*iValeurA) # => 'HelloHelloHelloHello'
```

```
sMotA="4"
iValeurA=4
print (sMotA+iValeurA)
# => TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Les Bases

Les variables – les opérations

```
iValeurA=4  
iValeurA=iValeurA+2  
print(iValeurA) # => 6
```

```
iValeurA=4  
iValeurA+=2  
print(iValeurA) # => 6
```

```
iValeurA=4  
iValeurA-=2  
print(iValeurA) # => 2
```

```
iValeurA=4  
iValeurA*=2  
print(iValeurA) # => 8
```

```
iValeurA=4  
iValeurA/=2  
print(iValeurA) # => 4
```

Les Bases

Les variables – forcer un type de variable

```
sMotA="4"  
iValeurA=4  
print(sMotA+str(iValeurA)) # => 44  
print(int(sMotA)+iValeurA) # => 8  
print(int(sMotA)+float(iValeurA)) # => 8.0
```

- int() : conversion en entier
- float() : conversion en réel
- str() : conversion en chaîne de caractère

Les Bases

Les variables – cas particulier des booléens

```
bBoolA=True
bBoolB=False
iValeurA=4
print (iValeurA+bBoolA) # => 5
print (iValeurA+bBoolB) # => 4
print (bBoolA+bBoolA) # => 2
print (bBoolA*bBoolA) # => 1
```

True équivaut à 1
False équivaut à 0

```
iValeurA=-4
iValeurB=0
print (bool (iValeurA) ) # => True
print (bool (iValeurB) ) # => False
```

```
sMotA="Helloworld!"
sMotB=""
print (bool (sMotA) ) # => True
print (bool (sMotB) ) # => False
```

« Existence » vaut True
« Contenu vide » vaut False

Les Bases

Les variables – cas particulier des caractères spéciaux

L'anti-slash « \ » est un caractère d'échappement dans les chaînes de caractères.

```
#Problème de base
sMessage="Ceci est une "erreur" !"

#Solution cheap
sMessage='Ceci est une "erreur" !'

#Mauvaise solution
sMessage='C'est est une "erreur" !'

#Vraie solution
sMessage="C'est une \"erreur\" !"

#Symbole tabulation \t
sMessage="Un element\tUn autre element"

#Symbole saut à la ligne \n
sMessage="Une ligne\nUne seconde ligne"
```

Les Bases

Les variables – les méthodes

```
# coding: utf-8
"""Python3.6"""

sMessage="Helloworld!"
print(sMessage)                # => 'Helloworld!'
print(sMessage.upper())        # => 'HELLOWORLD!'
print(sMessage.count("o"))     # => 2
print(sMessage.replace("!", " :")) # => 'Helloworld :)'
```

- Les méthodes sont des fonctions associées spécifiquement à un objet (i.e un type de variable)
- Elles sont appelées par Variable.nomMethode(argument)
- Ici, les méthodes sont associées aux objets « string »
- Toutes les chaînes de caractères possèdent ces 3 méthodes
- Des variables de type « int » ou « float » ne les possèdent pas

dir(variable) permet de lister toutes les méthodes associées à une variable

Les structures de données

Les structures de données

Les tuples

```
nTuple=(4,5,"Helloworld!",42)

print(nTuple) # => (4,5,'Helloworld!',42)

print(nTuple[0]) # => 4
print(nTuple[1]) # => 5
print(nTuple[2]) # => 'Helloworld!'
print(nTuple[3]) # => 42
print(nTuple[4]) # => IndexError: tuple index out of range

print(nTuple[-1]) # => 42
print(nTuple[-2]) # => 'Helloworld!'
print(nTuple[-3]) # => 5
print(nTuple[-4]) # => 4
print(nTuple[-5]) # => IndexError: tuple index out of range

print(nTuple[0:2]) # => (4,5)
print(nTuple[1:3]) # => (5,'Helloworld!')
print(nTuple[0:3:2]) # => (4,'Helloworld!')

print(nTuple[0:864]) # => (4,5,'Helloworld!',42)
print(nTuple[-126:864]) # => (4,5,'Helloworld!',42)

print(nTuple[:2]) # => (4,5)
print(nTuple[2:]) # => ('Helloworld!',42)
print(nTuple[::-1]) # => (42,'Helloworld!',5,4)
```

Les structures de données

Les tuples

```
nTuple=(4,5,"Helloworld!",42)

print(nTuple.index(42))# => 3

print(len(nTuple))# => 4
print(sum(nTuple))
# => TypeError: unsupported operand type(s) for +: 'int' and 'str'
print(sum(nTuple[:2]))# => 9

nTuple[2]="Byebyeworld!"
# => TypeError: 'tuple' object does not support item assignment
```

Une fois défini, un tuple ne peut pas être modifié

Les structures de données

Les listes (/tableaux)

```
nTuple=(4,5,"Helloworld!",42)

tList=[4,5,"Helloworld!",42]

print(tList[2])# => 'Helloworld!'
print(tList[-4])# => 4
print(tList[1:3])# => [5,'Helloworld!']

tList[2]="Byebyeworld!"
print(tList[2])# => 'Byebyeworld!'

nNewTuple=tuple(tList)
print(nNewTuple)# => (4,5,'Byebyeworld!',42)

nNewList=list(nTuple)
print(nNewList)# => [4,5,'Helloworld!',42]
```

Les listes fonctionnent comme les tuples, tout en étant modifiable.
Il est possible de convertir une liste en tuple et réciproquement.

Les structures de données

Les listes (/tableaux)

```
tListA=[4,5,"Hello world!",42]
tListB=tListA

tListB[0]=225

print(tListA) # => [225,5,'Hello world! ',42]
print(tListB) # => [225,5,'Hello world! ',42]

#####
tListA=[4,5,"Hello world!",42]
tListB=list(tListA) # eq. tListA[:] ou tListA.copy()

tListB[0]=225

print(tListA) # => [225,5,'Hello world! ',42]
print(tListB) # => [4,5,'Hello world! ',42]
```

La variable ne stocke pas réellement la liste mais l'adresse mémoire de la liste.

Assigner une liste à une autre variable revient à lui assigner la même adresse mémoire. En conséquence, modifier l'un ou l'autre des variables revient à modifier le contenu de la même liste.

Les structures de données

Les listes (/tableaux)

```
tListA=[4,5,"Helloworld!",42]

#Ajouter un élément à la fin
tListA.append(56)
print(tListA) # => [4,5,'Helloworld! ',42,56]

#Allonger une liste
tListA.extend([4,5] # eq. tListA=tListA+[4,5]
print(tListA)) # => [4,5,'Helloworld! ',42,56,4,5]

#Insérer un élément à une position spécifique
tListA.insert(2,"a") # eq. tListA=tListA[0:2]+["a"]+tListA[2:]
print(tListA) # => [4,5,'a','Helloworld! ',42,56,4,5]

#Retirer le premier élément correspondant
tListA.remove(5)
print(tListA) # => [4,'a','Helloworld! ',42,56,4,5]

#Retirer (et assigner) un élément à une position spécifique
oItem=tListA.pop(5)
print(tListA) # => [4,'a','Helloworld! ',42,56,5]
print(oItem) # => 4

oItem=tListA.pop()
print(tListA) # => [4,'a','Helloworld! ',42,56]
print(oItem) # => 5
```


Les structures de données

Les listes (/tableaux)

```
tListA=[4,5,"Helloworld!",42]

#Obtenir l'index du premier élément
iIndex=tListA.index(42) # eq. tListA=tListA.index(42,0,len(tListA)-1)
print(iIndex) # => 3

#Compter le nombre d'occurrence d'un élément
iQuantity=tListA.count(4)
print(iQuantity) # => 1

#Inverser l'ordre
tListA.reverse() # eq. tListA=tListA[::-1]
print(tListA) # => [42,'Helloworld!',5,4]

tListB=[4,5,42,32]

#Trier les éléments
tListB.sort()
print(tListB) # => [4,5,32,42]
tListB.sort(reverse=True)
print(tListB) # => [42,32,5,4]
```

La méthode « .sort() » ne fonctionne qu'avec des éléments homogènes (numérique ou alphabétique)

Les structures de données

Les listes (/tableaux)

```
tListA=[4,5,"Helloworld!",42]

#Obtenir la taille d'une liste
print(len(tListA)) # => 4

#générez automatiquement un tableau de nombre...
#...jusqu'à une valeur non-incluse
print(list(range(4))) # => [0,1,2,3]

#...entre une valeur incluse et une valeur non-incluse
print(list(range(1,4))) # => [1,2,3]

#...entre une valeur incluse et une valeur non-incluse, toutes les n
valeurs
print(list(range(1,4,2))) # => [1,3]

#...représentant les index d'une liste
print(list(range(len(tListA)))) # => [0,1,2,3]
```

Les structures de données

Les dictionnaires

Les listes et les tuples structurent les objets selon un ordre.
Les dictionnaires structurent les objets selon une relation clef-valeur.

Les structures de données

Les dictionnaires

```
nTuple=(4,5,"Helloworld!",42)

tList=[4,5,"Helloworld!",42]

dDict={"item1":4,"item2":5,"item3":"Helloworld!","item4":42}

print (dDict[0]) # => KeyError: 0

print (dDict["item1"]) # => 4

print (dDict.keys())
# => dict_keys(['item1', 'item2', 'item3', 'item4'])
print (dDict.values())
# => dict_values([4, 5, 'Helloworld!', 42])

#Ajouter un item
dDict["item5"]=125
#Ecraser un item
dDict["item1"]=326
#Supprimer un item
del dDict["item3"]
```

Les structures de données

Les dictionnaires

```
nTuple=(4,5,"Helloworld!",42)

tList=[4,5,"Helloworld!",42]

dDict={"item1":4,"item2":5,"item3":"Helloworld!","item4":42}

dDict[tList]=nTuple # => TypeError: unhashable type: 'list'

dDict[nTuple]=tList
print(dDict)
# => {'item1': 4, 'item2': 5, 'item3': 'Helloworld!', 'item4': 42,
(4, 5, 'Helloworld!', 42): [4, 5, 'Helloworld!', 42]}

dAnotherDict={"itemA":28}
dDict[dAnotherDict]=True # => TypeError: unhashable type: 'dict'
```

Seuls les objets non modifiable peuvent servir de clef.
N'importe quel objet peut servir de valeur.

Les structures de données

Les ensembles

```
setA={4,5,"Helloworld!",42}

print(setA[4]) # => TypeError: 'set' object is not subscriptable

setA.add(44) # => {4,5,44,"Helloworld!",42}
setA.update([45,46,47]) # => {4,47,5,44,45,"Helloworld!",42,46}

setA.discard(45) # => {4,47,5,44,"Helloworld!",42,46}
setA.discard("something") # => {4,47,5,44,"Helloworld!",42,46}

setA.remove(46) # => {4,47,5,44,"Helloworld!",42}
setA.remove("something") # => KeyError: "something"

# Suppression de doublons
tListB=[4,5,4,5,5,4,122,4,122,122,4,826]
setB=set(tListB)

print(setB) # => {4,5,122,826}

tListB=list(setB)

print(tListB) # => [4,5,122,826]
```

Les structures de données

Les ensembles - opérations

```
setA={4,5,"Helloworld!",42}
setB={4,5,122,826}

#Union (Contenu setA + Contenu setB)
print(setA | setB) # => {4,5,122,826,"Helloworld!",42}

#Intersection (Contenu commun à setA et setB)
print(setA & setB) # => {4,5}

#Difference (Contenu spécifique au setA)
print(setA - setB) # => {"Helloworld!",42}

#Difference symétrique (Contenu spécifique à chaque set)
print(setA ^ setB) # => {"Helloworld!",42,122,826}
```

Articuler le code

Articuler le code

Les conditions

```
if True:
    print("C'est vrai")
elif False:
    print("C'est faux")
else:
    print("C'est impossible")
```

- Indentation
 - ➔ Structure le code (Humain et interpréteur)
 - ➔ Une tabulation OU des espaces (4)
 - ➔ Ne JAMAIS mélanger espaces et tabulations
 - ➔ Un bloc indenté ne doit jamais être vide
- Les éléments « elif » et « else » sont facultatifs
- Il n'y a pas de limites au nombre de elif qu'on peut enchaîner

Articuler le code

Les conditions

```
if True:
    print("C'est vrai")
elif False:
    print("C'est faux")
else:
    print("C'est impossible")
```

```
if True:
    # IndentationError: expected an indented block
else:
    print("C'est faux")
```

```
if True:
    pass # Opération nulle permettant d'éviter le crash
else:
    print("C'est faux")
```

Articuler le code

Les conditions – les comparaisons

#égalité

5==6 # False

#différence

5!=6 # True

#Strictement inférieur (« > » pour strictement supérieur)

5<6 # True

#Inférieur ou égal (« >= » pour supérieur ou égal)

5<=6 # True

#Fait partie de...

5 **in** [5,6,7,8,9] # True

#Ne fait pas partie de...

5 **not in** [5,6,7,8,9] # False

#N'est pas... (« inverser » le résultat)

not 5==6 # True

Articuler le code

Les conditions – les comparaisons

```
#Strictement inférieur... par rapport à l'ordre alphabétique  
"100"<"8" # True
```

```
#Comportement distinct entre int et str : pas de melange
```

```
"100"<8
```

```
# => TypeError: '<' not supported between instances of 'str' and  
'int'
```

Articuler le code

Les conditions – multiples conditions

```
dDict={"item1":4,"item2":5,"item3":"Helloworld!","item4":42}
```

```
#L'une et/ou l'autre condition
```

```
if dDict["item1"]<10 or dDict["item2"]<10:  
    print("C'est vrai")
```

```
#L'une et l'autre condition
```

```
if dDict["item1"]<10 and dDict["item2"]<10:  
    print("C'est vrai")
```

```
#L'une ou l'autre condition mais pas les deux
```

```
if (dDict["item1"]<10 and not dDict["item2"]<10) or \  
    (not dDict["item1"]<10 and dDict["item2"]<10):  
    print("C'est vrai")
```

```
#L'une ou l'autre condition mais pas les deux
```

```
if (dDict["item1"]<10) ^ (dDict["item2"]<10): # /\ parenthèses  
    print("C'est vrai")
```

```
#L'une et/ou l'autre condition
```

```
if dDict["item1"]<10 or dDict["item3"]<10:  
    print("C'est vrai") # => TypeError(..)
```

Toutes les comparaisons d'une ligne sont vérifiées simultanément
Si l'une d'entre elle contient une erreur, l'ensemble crash.

Articuler le code

Les boucles

```
tList=[1,2,3,4,5,6,7,8,9,10]
```

```
#Mauvaise idée
```

```
TList[0]+=1
```

```
TList[1]+=1
```

```
TList[2]+=1
```

```
TList[3]+=1
```

```
TList[4]+=1
```

```
TList[5]+=1
```

```
TList[6]+=1
```

```
TList[7]+=1
```

```
TList[8]+=1
```

```
tList[9]+=1
```

Articuler le code

Les boucles – Boucle for

```
tList=[1,2,3,4,5,6,7,8,9,10]
```

```
#Mauvaise idée
```

```
TList[0]+=1
```

```
TList[1]+=1
```

```
TList[2]+=1
```

```
TList[3]+=1
```

```
TList[4]+=1
```

```
TList[5]+=1
```

```
TList[6]+=1
```

```
TList[7]+=1
```

```
TList[8]+=1
```

```
tList[9]+=1
```

```
#Boucle for
```

```
for iIndex in range(len(tList)):
```

```
    tList[iIndex]+=1
```

Articuler le code

Les boucles – Boucle while

```
tList=[1,2,3,4,5,6,7,8,9,10]
```

```
#Mauvaise idée
```

```
TList[0]+=1
```

```
TList[1]+=1
```

```
TList[2]+=1
```

```
TList[3]+=1
```

```
TList[4]+=1
```

```
TList[5]+=1
```

```
TList[6]+=1
```

```
TList[7]+=1
```

```
TList[8]+=1
```

```
tList[9]+=1
```

```
#Boucle for
```

```
for iIndex in range(len(tList)):
```

```
    tList[iIndex]+=1
```

```
#Boucle while
```

```
iIndex=0
```

```
while iIndex<len(tList): #/>\ len(tList)==10
```

```
    tList[iIndex]+=1
```

```
    iIndex+=1 #/>\ ne JAMAIS oublier la variable, sinon boucle infini
```


Articuler le code

Les boucles – Boucle for ou while ?

- Boucle for
 - Fonctionne parfaitement sur les structures dénombrables
 - Gestion automatique de la variable liée à la boucle
 - ~~Ne permet pas de modifier ladite structure dénombrable~~
- Boucle while
 - Fonctionne pour les cas indénombrables
 - Gestion manuelle de la variable liée à la boucle
 - Permet de modifier à la volée la structure parcourue
 - Gérer la variable liée à la boucle correctement

TOUJOURS utiliser une boucle for, sauf si c'est rigoureusement impossible

Articuler le code

Les boucles - exemple

```
tList=[1,2,3,4,5,6,7,8,9,10] #supprimer les chiffres pairs
```

```
#Boucle for
for iValue in tList:
    if iValue%2==0:
        tList.remove(iValue)
# => [1, 3, 5, 7, 9]
```

```
#Boucle while
iIndex=0
while iIndex<len(tList):
    if tList[iIndex]%2==0:
        tList.remove(tList[iIndex])
    else:
        iIndex+=1
# => [1, 3, 5, 7, 9]
```

Articuler le code

Les boucles – comprehension list

```
tList=[1,2,3,4,5,6,7,8,9,10] #supprimer les chiffres pairs
```

```
#Boucle for
for iValue in tList:
    if iValue%2==0:
        tList.remove(iValue)
# => [1, 3, 5, 7, 9]
```

```
#Boucle while
iIndex=0
while iIndex<len(tList):
    if tList[iIndex]%2==0:
        tList.remove(tList[iIndex])
    else:
        iIndex+=1
# => [1, 3, 5, 7, 9]
```

```
#Comprehension list
tList=[ X for X in tList if not X%2==0 ]
```

Articuler le code

Les boucles – continue et break

```
tList=[1,2,3,4,5,6,7,8,9,10] #supprimer les chiffres pairs
```

```
#Boucle for
```

```
for iValue in tList:
```

```
    if iValue==6:
```

```
        continue #passer immédiatement au tour de boucle suivant
```

```
    if iValue%2==0:
```

```
        tList.remove(iValue)
```

```
# => [1, 3, 5, 6, 7, 9]
```

```
#Boucle for
```

```
for iValue in tList:
```

```
    if iValue==6:
```

```
        break #Quitter immédiatement le bloc indenté lié à la boucle
```

```
    if iValue%2==0:
```

```
        tList.remove(iValue)
```

```
# => [1, 3, 5, 6, 7, 8, 9, 10]
```

Articuler le code

La gestion des erreurs

Articuler le code

La gestion des erreurs

```
dDict={"item1":4,"item2":5,"item3":"Helloworld!","item4":42}
```

```
#L'une et/ou l'autre condition
```

```
if dDict["item1"]<10 or dDict["item3"]<10:  
    print("C'est vrai") # => TypeError(..)
```

```
#Gestion de l'erreur, ignorer
```

```
try:  
    if dDict["item1"]<10 or dDict["item3"]<10:  
        print("C'est vrai")  
except TypeError:  
    print("C'est plus compliqué")
```

```
#Gestion de l'erreur, crasher quand même
```

```
try:  
    if dDict["item1"]<10 or dDict["item3"]<10:  
        print("C'est vrai")  
except TypeError:  
    print("C'est plus compliqué")  
    raise
```

Articuler le code

La gestion des erreurs

```
dDict={"item1":4,"item2":5,"item3":"Helloworld!","item4":42}
```

```
#L'une et/ou l'autre condition
```

```
if dDict["item1"]<10 or dDict["item3"]<10:  
    print("C'est vrai") # => TypeError(..)
```

```
#Gestion de l'erreur
```

```
try:  
    if dDict["item1"]<10 or dDict["item3"]<10:  
        print("C'est vrai")  
except TypeError:  
    print("C'est plus compliqué")  
finally:  
    print("Afficher ceci quoiqu'il arrive")
```

Articuler le code

Les fonctions

Articuler le code

Les fonctions

#Définir une fonction

```
def Racine(iValue,iRacine):  
    print(iValue**(1/iRacine))
```

```
Racine(4,2) # => 2.0
```

```
Racine(8,3) # => 2.0
```

#Stocker la réponse

```
fResult=Racine(4,2) # => 2.0
```

```
print(fResult) # => None
```

```
print(Racine(4,2))
```

```
# => 2.0
```

```
# => None
```

#Redéfinir la fonction pour qu'elle donne une réponse

```
def Racine(iValue,iRacine):  
    fReponse=iValue**(1/iRacine)  
    print(fReponse)  
    return fReponse
```

```
fResult=Racine(4,2)
```

```
print(fResult) # => 2.0
```

Le code est lu ligne par ligne : une fonction doit être définie en amont de son appel

Articuler le code

Les fonctions – réponses multiples

```
#Définir une fonction à réponses multiples
```

```
def Racine(iValue,iRacine):  
    fReponse=iValue**(1/iRacine)  
    sCommentaire="Helloworld!"  
    return fReponse, sCommentaire
```

```
fResult, sMessage=Racine(4,2)
```

```
print(fResult) # => 2.0
```

```
print(sMessage) # => 'Helloworld!'
```

Articuler le code

Les fonctions – réponses multiples

```
#Définir une fonction
def Racine(iValue,iRacine):
    fReponse=iValue**(1/iRacine)
    return fReponse, sCommentaire

#Redéfinir la même fonction
def Racine(iValue,iRacine):
    return "Helloworld!"

fResult=Racine(4,2)
print(fResult) # => 'Helloworld!'
```

Le code est lu ligne par ligne : si deux fonctions ont le même nom, seule la dernière version lue écrase la précédente

Articuler le code

Les fonctions – valeurs par défaut

#Définir une fonction avec une valeur par défaut

```
def Racine(iValue,iRacine=1):  
    fReponse=iValue**(1/iRacine)  
    return fReponse
```

```
fResult=Racine(4,2)  
print(fResult) # => 2.0  
fResult=Racine(4)  
print(fResult) # => 4.0
```

#Définir une fonction avec des valeurs par défaut

```
def Racine(iValue=4,iRacine=1):  
    fReponse=iValue**(1/iRacine)  
    return fReponse
```

```
fResult=Racine()  
print(fResult) # => 4.0  
fResult=Racine(iRacine=2)  
print(fResult) # => 2.0
```

Une valeur par défaut rend un argument facultatif.

Il est possible de forcer la valeur d'un argument facultatif via son nom, APRES les valeurs obligatoires.

Articuler le code

Les fonctions – portée des variables

```
#Définir une fonction à réponses multiples
def Racine(iValue,iRacine):
    fReponse=iValue**(1/iRacine)
    sInDef="Helloworld!"
    print(sInMain)
    return fReponse

#Main
sInMain="Variable du code principal"
fResult=Racine(4,2) # => 'Variable du code principal'
print(fResult) # => 2.0
print(sInMain) # => 'Variable du code principal'
print(sInDef) # => 'NameError: name 'sInDef' is not defined'
```

Une variable définie dans le corps principal du script est accessible partout.

Pour la lisibilité, il est néanmoins préférable de la faire transiter vis les arguments d'une fonction.

Une variable définie dans une fonction n'est pas accessible en dehors de cette fonction.

Articuler le code

Les fonctions – les librairies

```
#importer une librairie de fonctions
import time

iTimeA=time.time()
print("Helloworld!") # => 'Helloworld!'
iTimeB=time.time()
iDeltaTime=iTimeB-iTimeA
print(iDeltaTime) # => 0.001325368881225586
```

- Importer les fonctions : import MaLibrairie
- Appeler une fonction : MaLibrairie.MaFonction()
- Possibilité d'utiliser un alias

```
#importer une librairie de fonctions
import time as Te

iTimeA=Te.time()
print("Helloworld!") # => 'Helloworld!'
iTimeB=Te.time()
iDeltaTime=iTimeB-iTimeA
print(iDeltaTime) # => 0.001325368881225586
```

Articuler le code

Les fonctions – les librairies

- time : librairie standard d'accès au temps et conversion
time.time() : stocker l'heure en nombre de secondes
- random : librairie standard de génération de nombres pseudo-aléatoires
random.randint(0,10) : génère une valeur aléatoire comprise entre 0 (inclus) et 10 (inclus)
- sys : librairie standard de paramètres et fonctions systèmes
sys.exit() : clure le processus python
- os : librairie standard d'interface pour le système d'exploitation
os.listdir(MonDossier) : génère une de l'ensemble des fichiers présent dans un dossier de l'ordinateur

Liste des librairies standards : <https://docs.python.org/fr/3/library/>
Les librairies non-standards doivent être installées séparément

Exercice

Coder un jeu de bataille

« On distribue les 52 cartes aux joueurs (peut se jouer à deux) qui les rassemblent en paquet devant eux.

Chacun tire la carte du dessus de son paquet et la pose sur la table.

Celui qui a la carte la plus forte ramasse les autres cartes.

L'as est la plus forte carte, puis roi, dame, valet, 10, etc.

Lorsque deux joueurs posent en même temps deux cartes de même valeur il y a "bataille". Lorsqu'il y a "bataille" les joueurs tirent la carte suivante et la posent, face cachée, sur la carte précédente. Puis ils tirent une deuxième carte qu'ils posent cette fois-ci face découverte et c'est cette dernière qui départagera les joueurs.

Le gagnant est celui qui remporte toutes les cartes. »

Exercice

Coder un jeu de bataille

Liste

« On distribue les 52 cartes aux joueurs (peut se jouer à deux) qui les rassemblent en paquet devant eux.

Variable

Chacun tire la carte du dessus de son paquet et la pose sur la table.

Comparaison

Celui qui a la carte la plus forte ramasse les autres cartes.

Dictionnaire

L'as est la plus forte carte, puis roi, dame, valet, 10, etc.

Cas particulier

Condition

Lorsque deux joueurs posent en même temps deux cartes de même valeur il y a "bataille". Lorsqu'il y a "bataille" les joueurs tirent la carte suivante et la posent, face cachée, sur la carte précédente. Puis ils tirent une deuxième carte qu'ils posent cette fois-ci face découverte et c'est cette dernière qui départagera les joueurs.

Boucle

Le gagnant est celui qui remporte toutes les cartes. »

Exercice

Coder un jeu de bataille

« On distribue les 52 cartes aux joueurs (peut se jouer à deux) qui les rassemblent en paquet devant eux.

Chacun tire la carte du dessus de son paquet et la pose sur la table.

Celui qui a la carte la plus forte ramasse les autres cartes.

L'as est la plus forte carte, ~~puis roi, dame, valet, 10, etc.~~ mais **perd face au 2.**

Lorsque deux joueurs posent en même temps deux cartes de même valeur il y a "bataille". Lorsqu'il y a "bataille" les joueurs tirent la carte suivante et la posent, face cachée, sur la carte précédente. Puis ils tirent une deuxième carte qu'ils posent cette fois-ci face découverte et c'est cette dernière qui départagera les joueurs.

Le gagnant est celui qui remporte toutes les cartes. »

Exercice

Coder un jeu de bataille

- Créer un paquet de 52 cartes.
- Le partager équitablement de façon aléatoire entre deux joueurs.
- Faire s'affronter les deux joueurs.
- A chaque tour, chaque joueur pioche sa première carte. Le vainqueur emporte le pli et le place sous sa pioche. En cas d'égalité, une bataille a lieu (ajout d'une carte face cachée puis d'une carte face visible) jusqu'à ce qu'un vainqueur l'emporte.
- Le premier joueur à ne pas pouvoir piocher de carte a perdu.
- Le programme doit afficher :
 - Le nombre du tour en cours
 - Le nom des cartes jouées par chaque joueur (ou le fait qu'ils jouent une carte face cachée)
 - Annoncer une bataille
 - Signaler qui emporte le pli
 - Donner le nom du vainqueur de la partie

Manipuler les fichiers

Manipuler les fichiers

Ouvrir, écrire, fermer

```
#Ouvrir un fichier
MonFichier=open("Helloworld.py", "w")

#Ecrire dans un fichier
MonFichier.write("print(\"Helloworld!\")\n")
MonFichier.write("print(\"Byebyeworld!\")\n")

#Fermer un fichier
MonFichier.close()
```

- L'option « w » ouvre un nouveau fichier. Si le fichier existe déjà, le contenu existant est effacé.
- L'option « a » ouvre un fichier existant et écrit à la suite du contenu. Si le fichier n'existe pas, un nouveau fichier est créé.
- Ne pas oublier de fermer un fichier ouvert

Contrairement à la fonction print, la méthode .write() n'implique par forcément un retour à la ligne. Il faut utiliser le symbole \n.

Manipuler les fichiers

Fermeture automatique

```
with open("Helloworld.py", "w") as MonFichier:  
    #Ecrire dans un fichier  
    MonFichier.write("print(\"Helloworld!\")\n")  
    MonFichier.write("print(\"Byebyeworld!\")\n")
```

En passant par la construction « with », le fichier est automatiquement fermé en fin d'instruction, même en cas d'erreurs dans le bloc d'indentation du with.

Manipuler les fichiers

Lecture d'un fichier

```
with open("Helloworld.py", "r") as MonFichier:  
    sAllLineInOneString=MonFichier.read()
```

```
with open("Helloworld.py", "r") as MonFichier:  
    tListOfLines=MonFichier.readlines()
```

```
with open("Helloworld.py", "r") as MonFichier:  
    for sLine in MonFichier:  
        print(sLine.strip())
```

- L'option « r » lit le contenu d'un fichier.
- La méthode .strip() permet d'éliminer les caractères invisibles en début et fin de lignes (\n, retour chariot windows, etc.).

Exercice

Coder un jeu de bataille

Étape bonus n°9

Sauver l'ensemble des informations s'affichant à l'écran dans un fichier « Replay.txt »

Exercice

Coder un jeu de bataille

Étape bonus n°10

Coder un second programme chargé de lire le fichier Replay.txt.

Le programme devra afficher :

- Le nombre de tours joués
- Le nombre de batailles livrées, par ordre de grandeur (1 pour une carte face cachée, 2 pour deux cartes face cachée, etc.)
- L'identité du vainqueur

Quelques points intéressants

Mettre rapidement en forme du texte

```
#Ecriture fastidieuse d'une chaîne de caractère complexe
sMaVariable="Helloworld"
iValeur=10
print("blablabla "+sMaVariable+" "+str(iValeur)+" blablabla")

#Ecriture moins fastidieuse
sMaVariable="Helloworld"
iValeur=10
print("blablabla {} {} blablabla".format(sMaVariable,iValeur))

#Changement rapide de position
print("blablabla {1} {0} blablabla".format(sMaVariable,iValeur))
```

- La méthode format permet de gérer automatiquement la conversion en string.
- Elle remplace le double symbole {} du texte par ses différents arguments, dans l'ordre si rien n'est précisé, ou selon l'index de l'argument indiqué entre accolades

Mettre rapidement en forme du texte

```
#Ecriture fastidieuse d'une liste
tMaListe=["Helloworld", "Helloworld", "Helloworld"]
sChaine=""
for sItem in tMaListe:
    if sChaine!="":
        sChaine+="\t"
    sChaine+=sItem
print(sChaine)

#Ecriture moins fastidieuse
tMaListe=["Helloworld", "Helloworld", "Helloworld"]
print("\t".join(tMaListe))
```

- La méthode `.join()` permet de concaténer les éléments d'une structures itérables, en intercalant une chaîne de caractère entre chaque élément.
- Il s'agit d'une concaténation : l'itérable ne doit donc contenir que des éléments de type string.

Interaction avec l'utilisateur

```
sInput=input("Entrez un mot\n")
print(sInput) # => Le mot écrit par l'utilisateur

sInput=int(input("Entrez un nombre\n"))
print(sInput) # => Le nombre écrit par l'utilisateur
```

- La réponse entrée par l'utilisateur est toujours assimilée à du texte (c'est-à-dire "42" au lieu de 42).
- C'est au programmeur de gérer l'ensemble des possibilités de « mauvaise réponse ».
- input() peut servir à observer le déroulement d'une boucle pas-à-pas en forçant le code à attendre une action humaine avant de passer au tour de boucle suivant.

Donner des arguments à un script python

```
import sys
```

```
sMessage=sys.argv[1]
```

```
print(sMessage)
```

```
(base) antoine@antoine-Latitude-5590:~$ python3 print.py Hello world
Hello world
```

```
(base) antoine@antoine-Latitude-5590:~$ python3 print.py Autre chose
Autre chose
```

```
(base) antoine@antoine-Latitude-5590:~$ python3 print.py 42
42
```

- L'attribut `.argv` permet d'obtenir la liste des arguments de la ligne de commande lançant le script.
- `sys.argv[0]` renvoie le nom du script

Avec `sys.argv`, l'ordre des arguments est figé et doit être respecté. Les valeurs numériques sont récupérés sous forme de texte ('1' au lieu de 1)

Programmation orienté objet

Programmation orienté objet

- Un “objet” un bloc cohérent de code qui possède ses propres variables et ses propres fonctions.
- En Python, « tout est objet » : le type string est un objet possédant une variable propre (son contenu) et des fonctions propres (les méthodes : `.count`, `.replace`, etc.)

Programmation orienté objet

```
#Definition d'un objet « voiture »
class Voiture:
    def __init__(self, sColor, sType) :
        self.color=sColor
        self.type=sType
        self.position=0
        self.speed=0

#Main
oVoitureA=Voiture("Bleue", "2CV")
oVoitureB=Voiture("Rouge", "Coccinelle")
print(oVoitureA.color) # => 'Bleue'
print(oVoitureB.speed) # => 0
```

Programmation orienté objet

```
#Definition d'un objet « voiture »
class Voiture:
    def __init__(self, sColor, sType) :
        self.color=sColor
        self.type=sType
        self.position=0
        self.speed=0

#Main
oVoitureA=Voiture("Bleue", "2CV")
oVoitureB=Voiture("Rouge", "Coccinelle")
print(oVoitureA.color) # => 'Bleue'
print(oVoitureB.speed) # => 0
```

```
#Definition d'un dictionnaire de voitures
dVoitures={
    "VoitureA":{"type":"2CV", "color":"Bleue",
                "position":0, "speed":0},
    "VoitureB":{"type":"Coccinelle", "color":"Rouge",
                "position":0, "speed":0},
}

print(dVoitures["VoitureA"]["color"]) # => 'Bleue'
print(dVoitures["VoitureB"]["speed"]) # => 0
```

Programmation orienté objet

```
#Definition d'un objet « voiture »
class Voiture:
    def __init__(self, sColor, sType) :
        self.color=sColor
        self.type=sType
        self.position=0
        self.speed=0

    def speedUp(self, iValue) :
        self.speed+=iValue

    def move(self) :
        self.position+=self.speed

#Main
oVoitureA=Voiture("Bleue", "2CV")
oVoitureA.speedUp(15)
print(oVoitureA.position) # => 0
oVoitureA.move()
print(oVoitureA.position) # => 15
oVoitureA.move()
print(oVoitureA.position) # => 30
```

Programmation orienté objet

```
#Definition d'un objet « voiture »
class Voiture:
    def __init__(self, sColor, sType) :
        self.color=sColor
        self.type=sType
        self.position=0
        self.speed=0

    def speedUp(self, iValue) :
        self.speed+=iValue

    def move(self) :
        self.position+=self.speed

#Main
oVoitureA=Voiture("Bleue", "2CV")
oVoitureA.driver("Oui-oui")
print(oVoitureA.driver) # => 'Oui-oui'
```

Python autorise la création d'attribut à la volée. C'est généralement une mauvaise idée.

Programmation orienté objet

#Definition d'un objet « voiture »

class Voiture:

```
def __init__(self, sColor, sType) :  
    self.color=sColor  
    self.type=sType  
    self.position=0  
    self.speed=0
```

```
def getSpeed(self) :  
    return self.speed
```

```
def setSpeed(self, iValue) :  
    self.speed=iValue
```

```
def speedUp(self, iValue) :  
    self.setSpeed(self.getSpeed()+iValue)
```

```
def getPosition(self) :  
    return self.position
```

```
def setPosition(self, iValue) :  
    self.position=iValue
```

```
def move(self) :  
    self.setPosition(self.getPosition()+self.getSpeed())
```

Il est recommandé de passer par des fonctions dédiées (set et get) pour accéder et modifier les attributs d'un objet.

Programmation orienté objet

Afficher un objet

```
#Definition d'un objet « voiture »
class Voiture:
    def __init__(self, sColor, sType) :
        self.color=sColor
        self.type=sType
        self.position=0
        self.speed=0

    def __repr__(self):
        return self.type+" "+self.color

#Main
oVoitureA=Voiture("Bleue", "2CV")
print(oVoitureA) # => '2CV Bleue'
```

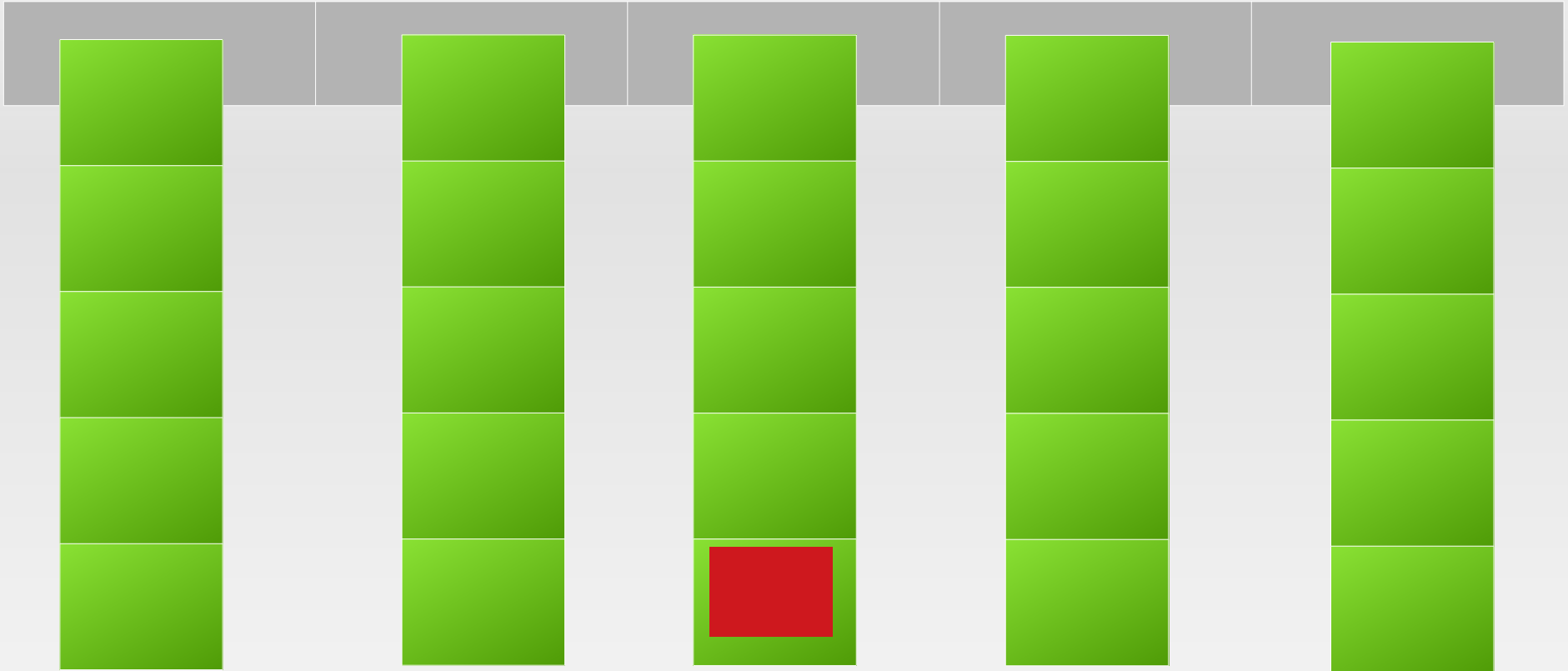
Exercice

Coder un simulateur de pandémie

- Créer un tableau de 10x10, contenant un objet Case. A son initialisation, chaque Case peut recevoir le statut vide ou sain (densité de population au choix).
- Afficher les lignes du tableau. Les Cases vides n'affichent rien, les Cases saines affichent « O ».
- Choisir aléatoirement une Case saine du tableau et lui donner le statut malade (« S »).
- A chaque tour de boucle, une Case malade a une probabilité d'obtenir le statut mort (« X »). Si elle ne meurt pas, une Case malade a une probabilité de devenir immune (« I »).
Une Case saine en contact (diagonale incluse) le tour précédent d'une Case malade a une probabilité de devenir malade.
- La simulation s'arrête lorsqu'il n'y a plus de malade.
- Afficher à chaque tour les chiffres de la population non-malade, malade et morte

Exercice

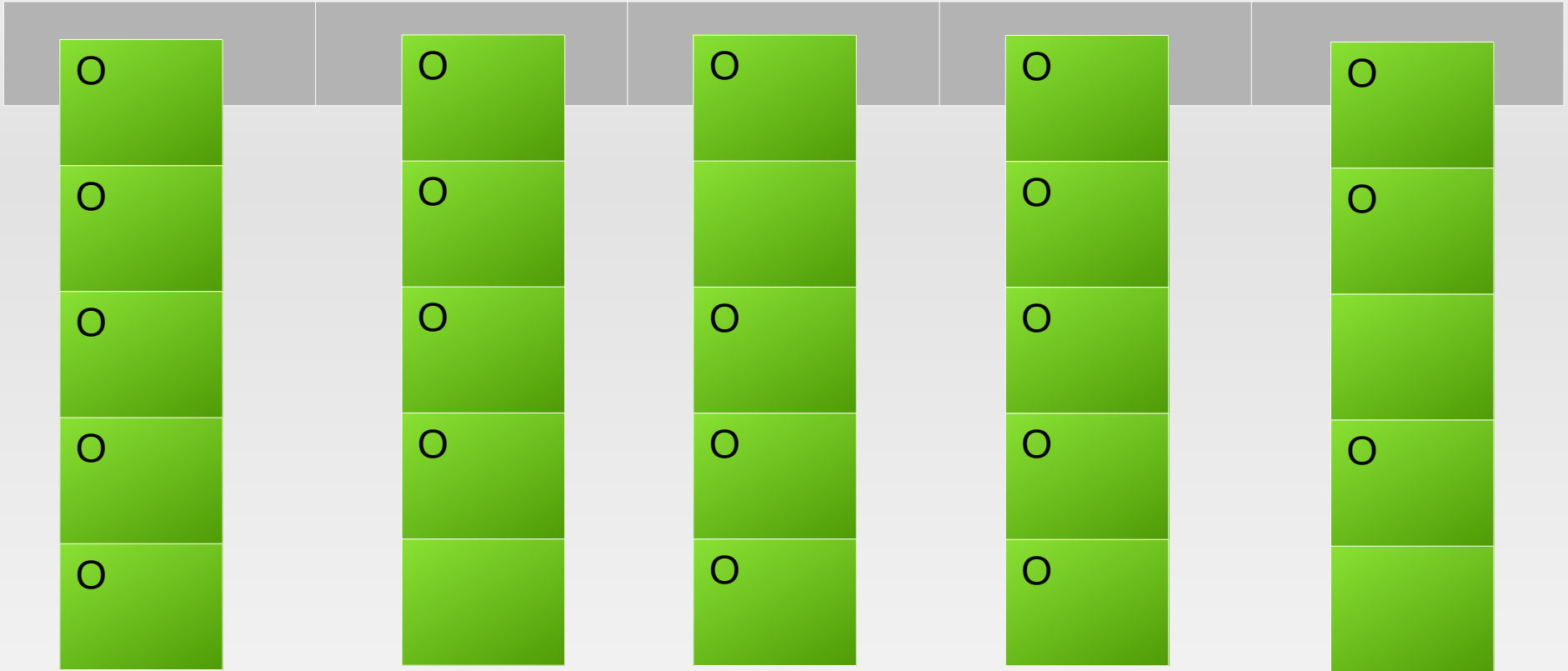
Coder un simulateur de pandémie



- Création d'un tableau à deux dimensions (une liste contenant une liste à chaque index)
- Rappel : Pour accéder à la case rouge : `listPrincipale[2][5]`

Exercice

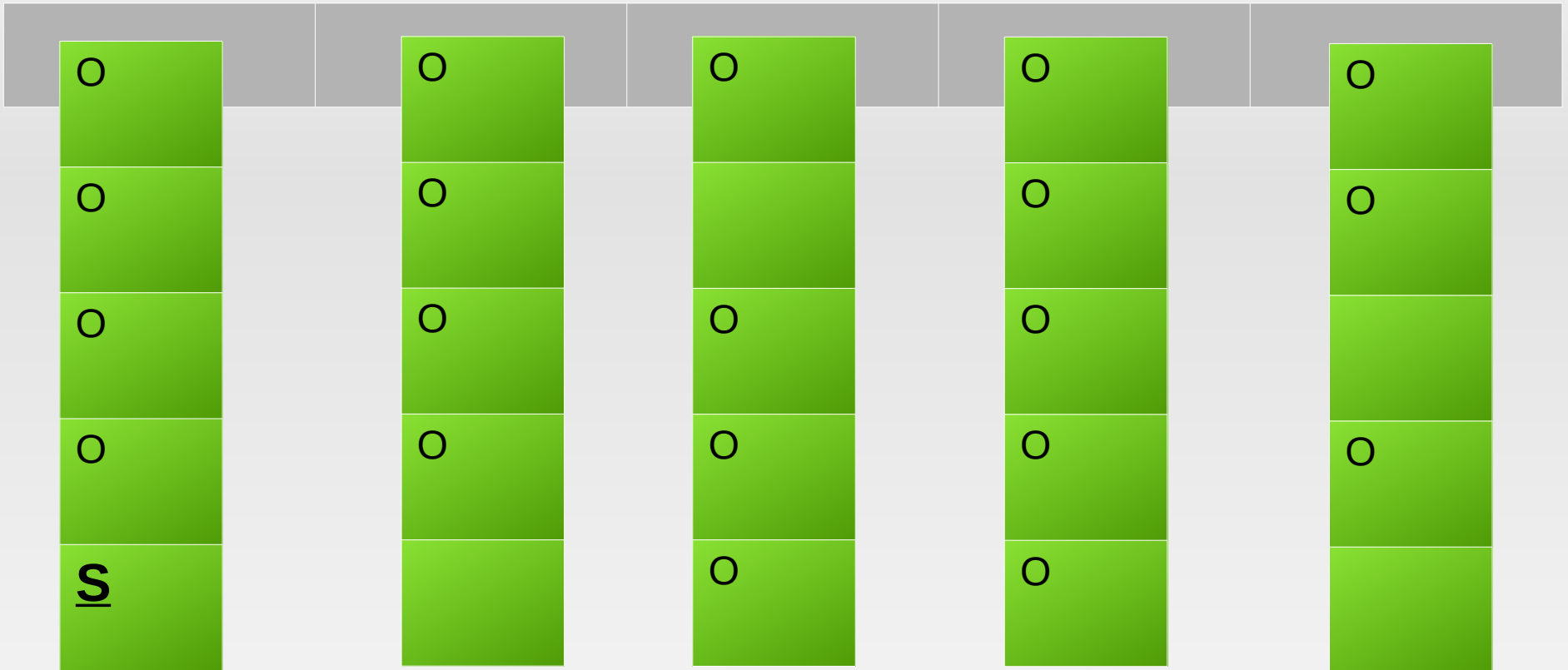
Coder un simulateur de pandémie



- Initialiser des objets Case, soit sain, soit vide.

Exercice

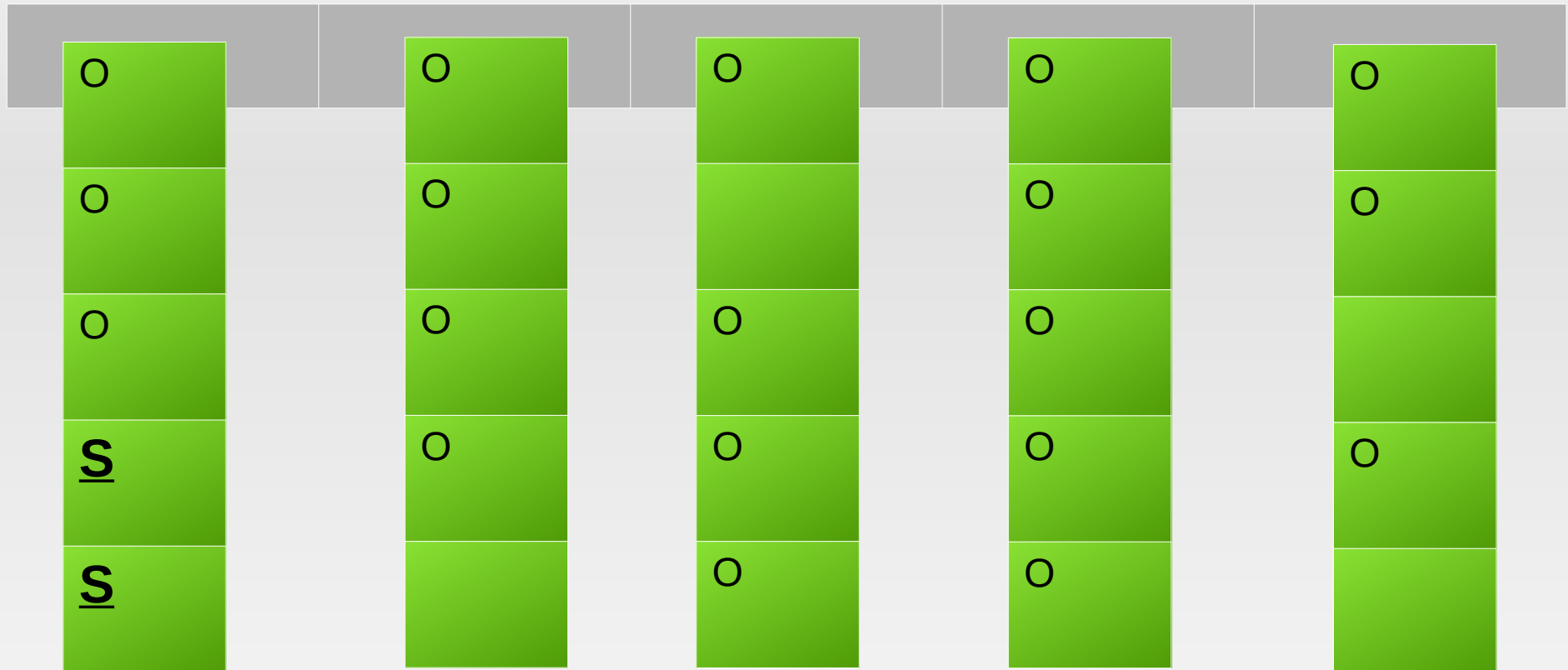
Coder un simulateur de pandémie



- Inoculer une case Sain au hasard → statut Malade

Exercice

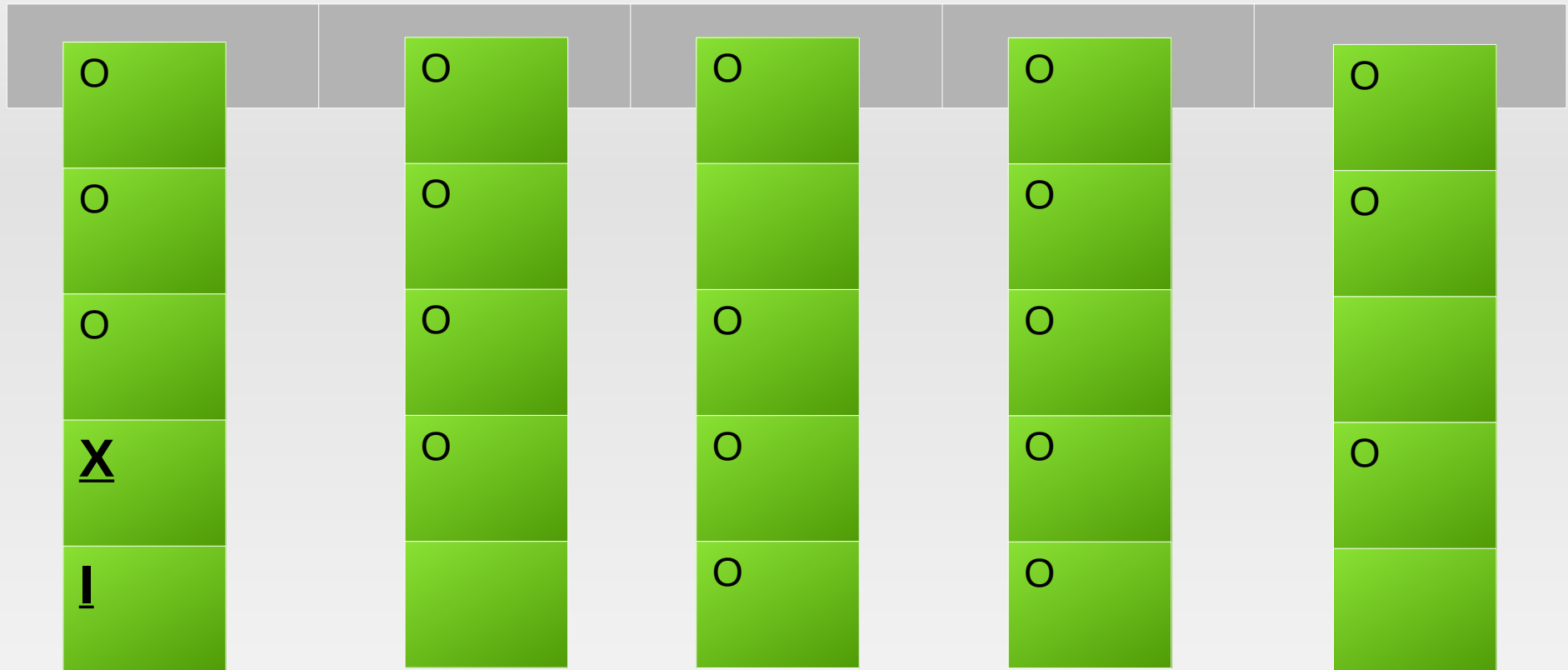
Coder un simulateur de pandémie



- Tour 1 :
La Case Malade (0,4) ne meurt ni ne guérit
Les Cases à proximité testent et la Case (0,3) devient Malade

Exercice

Coder un simulateur de pandémie



- Tour 2 :
 - La Case Malade (0,4) devient Immunisé
 - La Case Malade (0,3) devient Mort
 - Les Cases à proximité testent et aucune ne deviennent Malade
- Il n'y a plus de Case Malade, la simulation se termine

Exercice

Coder un simulateur de pandémie

- Créer un tableau de 10x10, contenant un objet Case. A son initialisation, chaque Case peut recevoir le statut vide ou sain (densité de population au choix).
- Afficher les lignes du tableau. Les Cases vides n'affichent rien, les Cases saines affichent « O ».
- Choisir aléatoirement une Case saine du tableau et lui donner le statut malade (« S »).
- A chaque tour de boucle, une Case malade a une probabilité d'obtenir le statut mort (« X »). Si elle ne meurt pas, une Case malade a une probabilité de devenir immune (« I »).
Une Case saine en contact (diagonale incluse) d'une Case malade a une probabilité de devenir malade.
- La simulation s'arrête lorsqu'il n'y a plus de malade.
- Afficher à chaque tour les chiffres de la population non-malade, malade et morte