

Image: pngtree

Option IA & Programmation

Cycle 1: Base de l'algorithmie

Alexandre Mazel

Année 2021-2022

alexandre.zelma@gmail.com

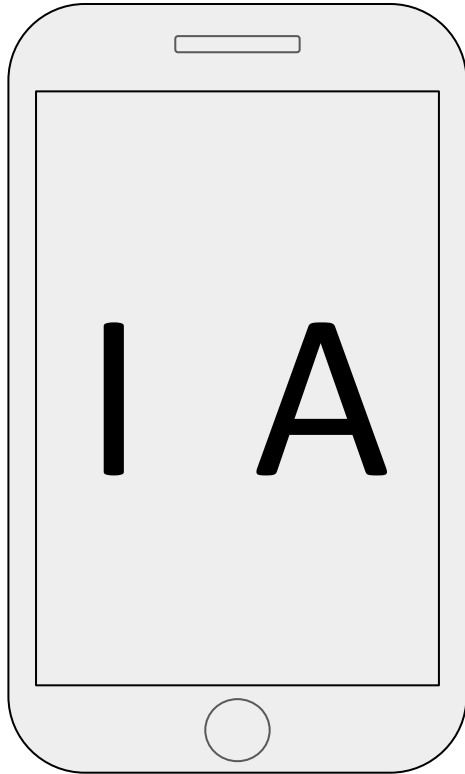
But de cette option



I A

- Comprendre ce qui se cache derrière les fameuses lettres I et A
- Démystifier cette technologie par la pratique
- Réfléchir à:
 - Ce que cela fait,
 - ne fait pas (encore?),
 - les risques humains que cela pourrait engendrer.

Programme de l'année



Découvrir et expérimenter les bases de l'IA et de sa programmation.

5 cycles:

1. Bases de l'algorithmie
2. Architecture d'un projet
3. Intelligence Artificielle
4. Expérimentation et amélioration autour d'un projet par petits groupes
5. Continuation du projet

Contenu du cycle 1

Cycle 1: Base de l'algorithmie

- Programme et variable
- Test et logique binaire
- Boucle
- Fonction
- Fichier
- Test unitaire

Déroulé des séances

- 5 minutes d'ouverture
- 20 minutes de présentations et discussions
- 1h de pratique
- 5 minutes de débrief

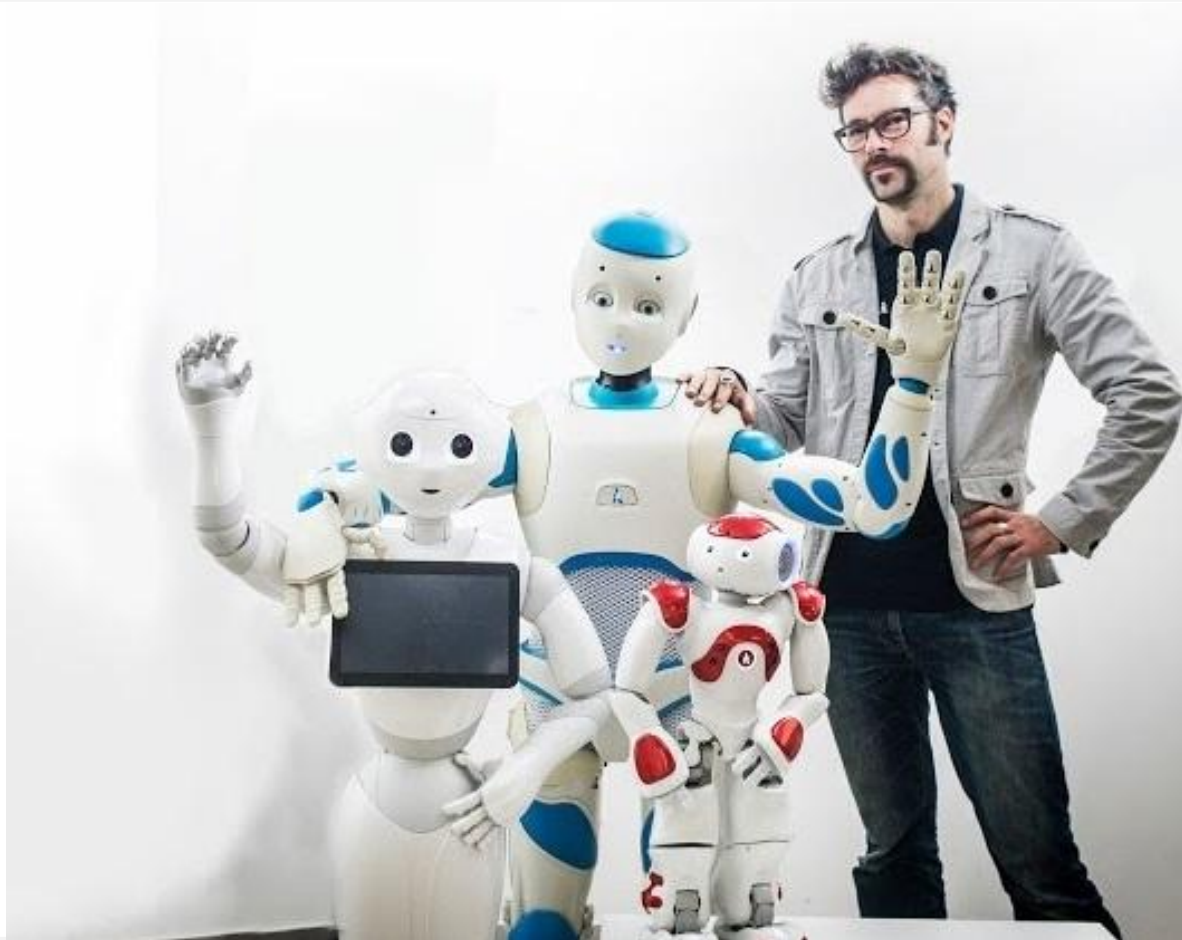
Mais c'est qui Alexandre Mazel ?



Directeur Innovation dans la robotique

- Diplomes:
 - Analyste-programmeur
 - Expert en systèmes informatiques
 - IA et reconnaissances de formes
- Jobs:
 - Moteur de vie artificielle (PNJ & CPU)
 - Développement de jeux vidéos
 - Développement de robots humanoïdes

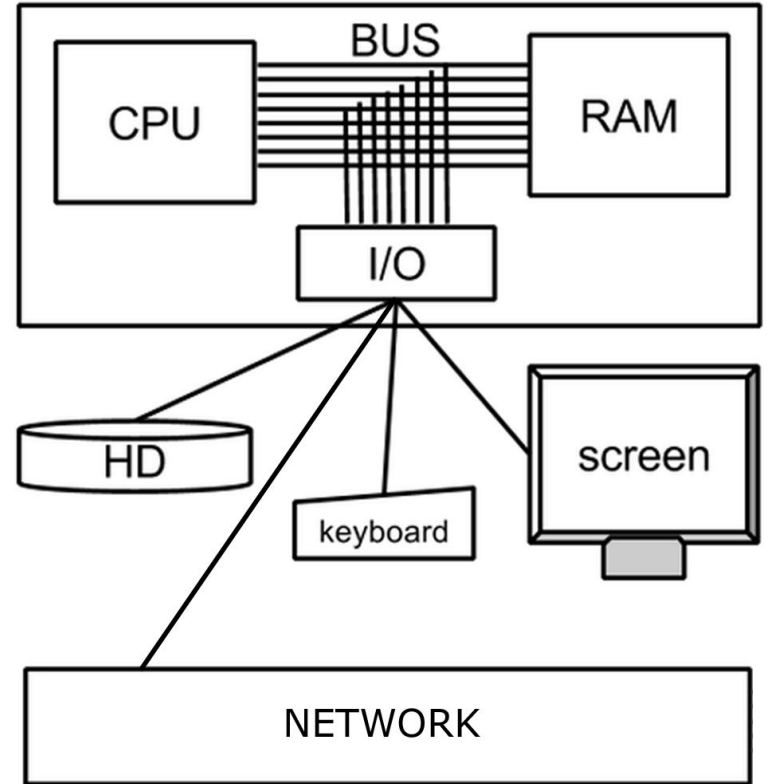
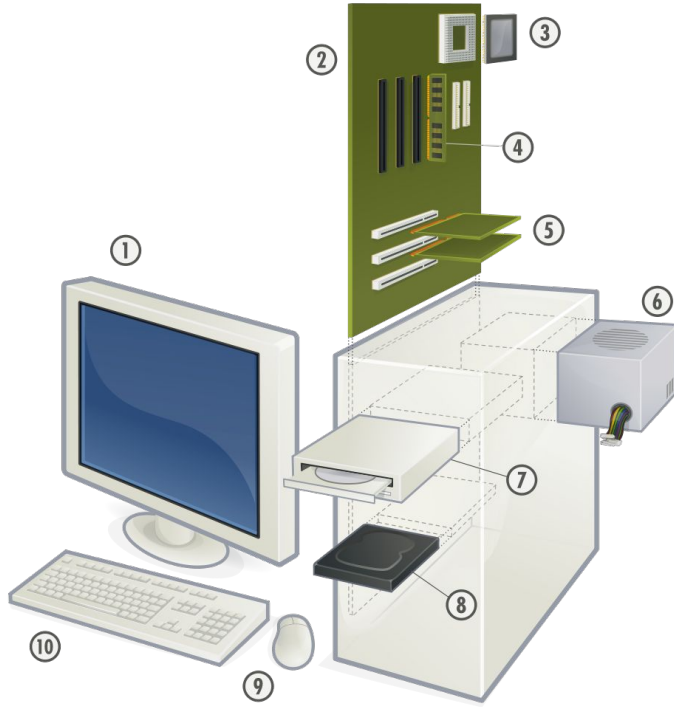
Mes robots



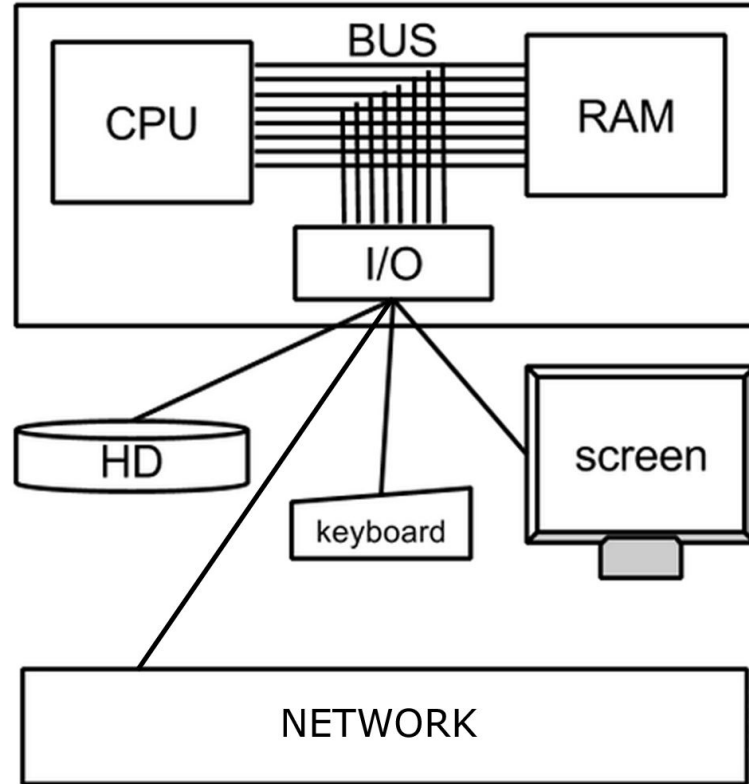
Partie 1

Programmes et variables

Un ordinateur



Le programme dans l'ordinateur



Un programme c'est comme une recette

1. Remplir une casserole d'eau salée
2. Faire bouillir
3. Ajouter les pates
4. Attendre 10 minutes
5. Egoutter
6. Servir

L'ordinateur est un idiot



credits: <https://www.youtube.com/watch?v=Ct-IOOUqmyY>

La magie n'existe pas



Une recette ultra précise

1. Remplir à moitié une grosse casserole d'eau salée
2. Faire bouillir:
 - Allumer un rond électrique
 - Placer la casserole sur le rond allumé
 - Attendre que l'eau fasse des bulles de 3cms
3. Ajouter les pâtes dans la casserole
4. Attendre 10 minutes en vérifiant que ca ne déborde pas
5. Egoutter
 - Mettre une passoire dans l'évier
 - Verser le contenu de la casserole dans la passoire
6. Servir
7. Éteindre le feu !

Un programme informatique

```
fonction Multiplie( a, b )  
  
    resultat = 0  
  
    compteur = 0  
  
    tant que compteur < b faire:  
        resultat = resultat + a  
        compteur = compteur + 1  
  
    donner le resultat
```

Un programme informatique

```
x = 1
tant que x < 5:
    afficher(x)
    x = x + 1
```

1
2
3
4

Les variables

Wikipedia:

les **variables** sont des éléments qui associent un nom (l'identifiant) à une valeur.

La valeur peut être de nature différente : nombre, texte, etc.

Les **variables** sont physiquement implantées dans la mémoire du système programmé (ordinateur, carte microprocesseur, etc.)

Les variables

Exemple de variable contenant un entier:

```
x = 3
```

```
x = x + 1
```

```
# x contient maintenant la valeur 4
```

```
print(x) # la valeur de x va être affichée: 4
```

Les variables

Exemple de variable contenant un décimal:

```
y = 3.5
```

```
y = y * 2
```

```
# y contient maintenant la valeur 7.00000
```

```
print(y) # la valeur de y va être affiché:
```

```
7.00000
```

Les variables

Exemple de variable contenant une chaîne:

```
n = "toto"
```

```
# n contient maintenant la valeur "toto"
```

```
print(n) # la valeur de n va être affichée:
```

```
"toto"
```

Les variables

Quelques règles de nommage des variables:

- Peut contenir des majuscules, minuscules et chiffres.
- Ne peut pas commencer par un chiffre.
- Ne peut pas contenir de signes (+/./;”...”...) ni d’espaces, le seul autorisé est ‘_’
- “sensible a la casse”: Les majuscules comptent:

```
a = 2
```

```
A = 30
```

```
# a vaut 2 mais A vaut 30
```

Les variables

On peut nommer les variables comme on veut, mais c'est plus simple d'être explicite pour se souvenir du rôle de la variable.

Ne pas faire:

```
nombre1 = 2
```

```
nombre2 = 4
```

```
nombre3 = 100
```

```
nombre = "alexandre"
```

Un programme informatique

```
x = 1  
tant que x < 5:  
    afficher(x)  
    x = x + 1
```



Ceci est un bloc d'instructions.

```
1  
2  
3  
4
```

Le langage python

Language open source et très actif.

Recommandé pour l'intelligence artificielle.



Particularités:

- Epuré.
- Indentation importante, sert à séparer les blocs d'instructions !
- Variables faciles à déclarer et type changeable à la volée.
- Interactif et auto inspectable.

Un programme en python

```
x = 1
while x < 5:
    print(x)
    x = x + 1
```

1
2
3
4

Un programme en python et en C

```
x = 1
while x < 5:
    print(x)
    x = x + 1
```

1
2
3
4

Python

```
int x = 1;
while( x < 5 )
{
    print(x);
    x = x + 1;
}
```

1
2
3
4

C/C++

Un autre programme en python

```
for x in range(1,5)  
    print(x)
```

1

2

3

4

Un programme en python

```
def multiply( a, b ):
```

```
    result = 0
```

```
    count = 0
```

```
    while count < b:
```

```
        result += a
```

```
        count += 1
```

```
    return result
```

```
# utilisation:
```

```
print( multiply(2,3) )
```

Un autre programme en python

```
def Multiply( a, b ):  
    result = 0  
    for i in range(b):  
        result += a  
    return result
```

```
# utilisation:  
print( multiply(2,3) )
```

Comment ça fonctionne ?

```
x = 1
while x < 5:
    print(x)
    x = x + 1
```

Faisons tourner le programme à la main !

1
2
3
4

Modifications

```
x = 1
while x < 5:
    print(x)
    x = x + 1
```

1
2
3
4

Et si je voulais afficher de 1 à 100 ?

Que les nombres pairs?

De 100 à 1 ?

Des questions ?

???



credits: facebook

Mon premier programme

A large, stylized, hand-drawn word "Hello!" in red ink. The letters are thick and expressive, with a casual, sketchy feel. The exclamation mark is also drawn in the same style.

- Créer un dossier pour ranger ses fichiers sources, par exemple: "OptionIA"
- Lancer EduPython
- Créer un nouveau fichier
- Sauver le fichier sous le nom "hello.py"
- Afficher un message avec la commande print, p.Ex: "hello"
- Lancer le programme

hello.py: la solution

```
print( "Bonjour humain!" )
```

Je compte

1

2

3

4

.

.

.

- Créer un nouveau fichier
- Sauver le fichier sous le nom “comptage.py”
- Afficher les nombres de 1 à 10
- Puis de 10 à 1
- Puis 10,20,30,...,100

comptage.py: la solution

```
x = 1
while x <= 10:
    print(x)
    x = x + 1
```

Les variables

Exemple de variable contenant un tableau:

```
a = [1, 2, 3]
```

```
prenoms = ["Alex", "Elsa", "Gaia"]
```

```
prenoms_et_age = [ ["Alex", 13], ["Gaia", 12] ]
```

le premier index est le **0** !!!

```
print(a[0])           # => 1
```

```
print(prenoms[2])     # => "Gaia"
```

Les tableaux

```
# comment modifier un tableau
```

```
a = [1, 8, 3]
```

```
a[1] = 2          # a vaut [1, 2, 3]
```

```
a[3] = 4          # => out of range (hors du tableau)
```

```
print(len(a))     # => 3
```

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

```
print(len(a))     # => 4
```

Dire bonjour à tout le monde



- Faire un programme qui salue 3 personnes à la suite en utilisant un tableau contenant des prénoms.
 - P.ex:
Bonjour Alex
Bonjour Toto
Bonjour Steph
Indice: "a" + "b" => "ab"
- Level up: saluer et donner les ages.
Hint: convertir un nombre en chaine: $s = \text{str}(n)$
- Level++: salue, et donne la différence d'âges par rapport au précédent.

Bonjour_tout_le_monde1.py

```
people = ["Patrick", "Momo", "Eric"]  
i = 0  
while i < 3:  
    print("Bonjour " + people[i] )  
    i += 1 # (equivaut à i = i + 1)
```

Bonjour_tout_le_monde1b.py

```
people = ["Patrick", "Momo", "Eric"]  
i = 0  
while i < len(people):  
    print("Bonjour " + people[i] )  
    i += 1
```

Bonjour_tout_le_monde2.py

```
people = ["Patrick",30,"Momo",20,"Eric",40]
i = 0
while i < len(people):
    print("Bonjour " + people[i] )
    print("Tu as " + str(people[i+1]) + " ans!" )
    i += 2
```

Bonjour_tout_le_monde2b.py

```
people = ["Patrick",30,"Momo",20,"Eric",40]
i = 0
while i < len(people)/2:
    print("Bonjour " + people[i*2] )
    print("Tu as " + str(people[i*2+1]) + " ans!" )
    i += 1
```

Bonjour_tout_le_monde2c.py

```
people = [["Patrick",30],["Momo",20],["Eric",40]]
i = 0
while i < len(people):
    print("Bonjour " + people[i][0] )
    print("Tu as " + str(people[i][1]) + " ans!" )
    i += 1
```

Les tableaux a n dimensions

Exemple de variable contenant un tableau hétérogènes:

```
a = [ [1,2,3], "Alex", 18, [100], [4,5,6] ]
```

```
prenoms_et_age = [ ["Alex",13], ["Gaia",12] ]
```

```
bbb = [[[10,20]]]
```

le premier index est toujours le 0 !!!

<code>len(a)</code>	# => 5	<code>a[2]</code>	=> 18
<code>a[0]</code>	# => [1,2,3]	<code>prenoms_et_age[0][1]</code>	=> 13
<code>len(a[0])</code>	# => 3	<code>prenoms_et_age[1][1]</code>	=> 12
<code>a[0][0]</code>	# => 1	<code>bbb[0][0][1]</code>	=> 20
<code>a[1]</code>	# => "Alex"	<code>len(bbb[0][0])</code>	=> 2

Bonjour_tout_le_monde2b.py

```
people = [  
    ["Patrick", 30],  
    ["Momo", 20],  
    ["Eric", 40]  
]  
  
i = 0  
while i < len(people):  
    print("Bonjour " + people[i][0] )  
    print("Tu as " + str(people[i][1]) + " ans!" )  
    i += 1
```

The diagram illustrates list indexing. A yellow box contains the expression `people[x][y]`. Blue arrows show the mapping: one arrow points from `x` to the first element of the list (e.g., "Patrick"), and another arrow points from `y` to the second element (e.g., 30) within the list structure.

Bonjour_tout_le_monde2c.py

```
people = [  
    "Patrick", 30,  
    "Momo", 20,  
    "Eric", 40  
]  
  
i = 0  
while i < len(people):  
    print("Bonjour " + people[i*2] )  
    print("Tu as " + str(people[i*2+1]) + " ans!" )  
    i += 1
```


Bonjour_tout_le_monde3.py

```
people = [["Patrick",30],["Momo",20],["Eric",40]]
i = 0
while i < len(people):
    print("Bonjour " + people[i][0] )
    print("Tu as " + people[i][1] + " ans!" )
    if i > 0:
        diff = people[i][1] - people[i-1][1]
        print("Tu as " + str(diff) + " ans de plus que le
précédent.")
    i += 1
```

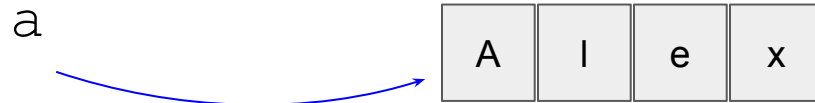
Bonjour_tout_le_monde3b.py

```
people = [["Patrick",30],["Momo",20],["Eric",40]]
i = 0
while i < len(people):
    iOther = ( i - 1 ) % len(people)
    print("Bonjour " + people[i][0] )
    print("Tu as " + people[i][1] + " ans!" )
    diff = people[i][1] - people[iOther][1]
    print("C'est " + str(diff) + " ans de plus que " + people[iOther][0].")
    i += 1
```

Indices de chaînes de caractères

```
a = "Alex"
```

En mémoire a est en fait un tableau homogène de caractères:



le premier index est toujours le 0 !!!

<code>len(a)</code>	# => 4	<code>a[:-1]</code>	=> 'Ale'
<code>a[2]</code>	# => 'e'	<code>a[0] + a[1:]</code>	=> 'Alex'
<code>a[1:3]</code>	# => 'le'	<code>a[0]*3 + a[1:]</code>	=> 'AAAlex'
<code>a[1:]</code>	# => 'lex'	<code>a[:]</code>	=> 'Alex'
<code>a[:2]</code>	# => 'Al'	<code>a[::2]</code>	=> 'Ae'
<code>a[-1]</code>	# => 'x'	<code>a[::-1]</code>	=> 'xelA'
<code>a[-2:]</code>	# => 'ex'	<code>a[0] = "b"</code>	impossible de changer un caractere d'une chaine.

Super CaPiTaLiSeUr



Faire un programme qui met une lettre sur deux en majuscule.

- Créer une variable contenant une chaîne de caractère avec plusieurs mots
- Parcourir la chaîne
- En créant une nouvelle variable avec la chaîne “capitalisée”.

```
"Alex".lower()    # => "alex"
```

```
"Alex".upper()    # => "ALEX"
```

```
s.lower()    # le texte dans s est en minuscule
```

Variante: Mettre une majuscule au début de chaque mot

capitaliseur.py

```
s = "le petit chaperon rouge est dans la foret."

dest = ""
n = 0
while n < len(s):
    if n % 2: # <==> (n%2) == 1
        dest = dest + s[n]
    else:
        dest = dest+s[n].upper()
    n += 1
```

Nommage des variables

On peut nommer les variables comme on veut...

... Mais c'est pratique d'avoir des règles, comme une hygiène de vie.

CamelCase

```
aAgeAndSize = [20,180]  
x = 3  
nCount = 18  
def getNumber():  
    return 3
```

Lower_with_underscore "lwu"

```
age_and_size = [20,180]  
x = 3  
count = 18  
def get_number():  
    return 3
```

Nommage des variables

La convention qu'on essaiera de tenir cette année se nomme la "PEP8".

En voici un exemple:

```
class SchoolStudent:                                # nom de classe en camel case

    def __init__( student_name )                    # nom de méthode fonction et parametre en lwu

        self.name = student_name                    # variable en lwu

    def compute_marks( self ):

        ...

# Nommage des choses en utilisant l'anglais (pas d'accent a gérer)
```

Les tests

Exemple de test en python:

```
if x > 100:

    print("x est vachement grand")

elif x < -100:

    print("x est carrément petit")

else:

    print("x est assez normal en fait")
```


Les opérateurs booléen

a	b	a or b
False	False	False
False	True	True
True	False	True
True	True	True

a	b	a and b
False	False	False
False	True	False
True	False	False
True	True	True

Les tests

Exemple de test en python:

```
if x > -100 and x < 100:  
    print("x est assez normal en fait")  
  
# python est parfait pour les paresseux:  
if -100 < x < 100:  
    print("x est assez normal en fait")
```

Les tests

Attention au parenthèse:

```
if x > -100 and y > 100 or z > 1000:
```

```
    print("je sais pas trop")
```

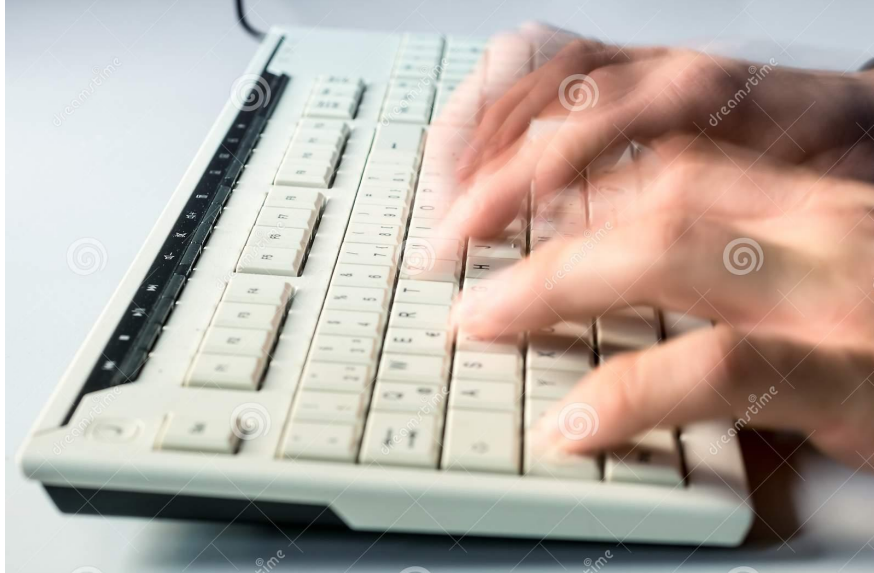
```
if x > -100 and ( y > 100 or z > 1000 ):
```

```
    print("l'un")
```

```
if (x > -100 and y > 100) or z > 1000:
```

```
    print("ou l'autre")
```

Fast and Keyrious



Faire un programme qui demande 5 fois à l'utilisateur de taper un mot au hasard.

Le programme chronomètre le temps total, puis affiche le temps moyen par caractère tapé.

Fonction utile:

- `random.randint(min,max_inclus)`
=> un nombre dans `[min,max_inclus]`
- `input()` et `input("prompt:")`
- `time.time()` => heure actuelle en sec

Pour les plus forts: au lieu de demander un mot, demander le résultat d'un calcul (table de multiplication) ou la traduction du mot dans une autre langue.

Keyrious.py (draft)

```
words = [  
    "banana",  
    "apple",  
    "citron",  
    "piano",  
    "panier"  
]  
  
time_begin = time.time()  
while qqchose:  
    idx_word = random.randint(0,qqchose)  
    word = words[qqchose - idx_word - 1]  
    user_word = input("Tape le mot " + word + " ")  
    ...  
time_end = time.time()
```

keyrious.py

```
import time
import random
words = [
    "banana",
    "apple",
    "citron",
    "piano",
    "panier"
]

time_begin = time.time()
count = 0
while count < 5:
    idx_word = random.randint(0, len(words)-1)
    word = words[idx_word]
    user_word = input("Tape le mot " + word + ":\n" )
    while word != user_word:
        user_word = input("reessaye de taper le mot " + word )
    count += 1
time_end = time.time()

print("tu as mis %.3f secondes" % (time_end-time_begin))
```

Mes questions

- Ca va trop vite ?
- Vous vous attendiez à ce contenu ?
- Plus de pratiques ?
- Plus de blagues ?
- Plus d'images ?
- Plus de formules mathématiques compliquées ?
- Plus d'IA comme dans les films de science-fictions ?

Les fichiers

LE seul moyen de garder une information entre 2 exécutions de programme.
Et même si la machine est redémarré !

Le fichier peut être en local (disque dur ou clé usb) ou distant (autre ordinateur proche ou sur le web).

Récupérer un objet mappant un fichier sur le disque:

```
file = open(nom_du_fichier, mode_d_ouverture)
```

```
nom_du_fichier: "t.txt", "t", "c:\\tutu", "c:/toto.txt"
```

```
mode_d_ouverture:
```

- "r" pour le lire.
- "w" pour l'écrire. CELA va en créer un nouveau cad effacer un potentiel fichier ayant le même nom.
- "a": pour ajouter a la fin du fichier (append)

Les fichiers

```
# usage des fichiers:
# ecriture
file = open("toto.txt", "w")
file.write( "Salut toto!")
file.close()

# lecture
file = open("toto.txt", "r")
s = file.read()
file.close()
print(s)
```

autour des fichiers

```
# récupérer la liste des fichiers d'un dossier
files = os.listdir("c:\\")
```

savoir si un fichier existe

```
is_exist =
os.path.isfile("toto.txt")
```

récupérer la taille d'un fichier

```
size = os.path.getsize("toto.txt")
```

Les fonctions: mais pourquoi ?



credits: ebay.fr

Les copier-coller c'est LE mal:
Avoir plus de 3 lignes identiques à
2 endroits, c'est moche.

Les fonctions

sans fonction

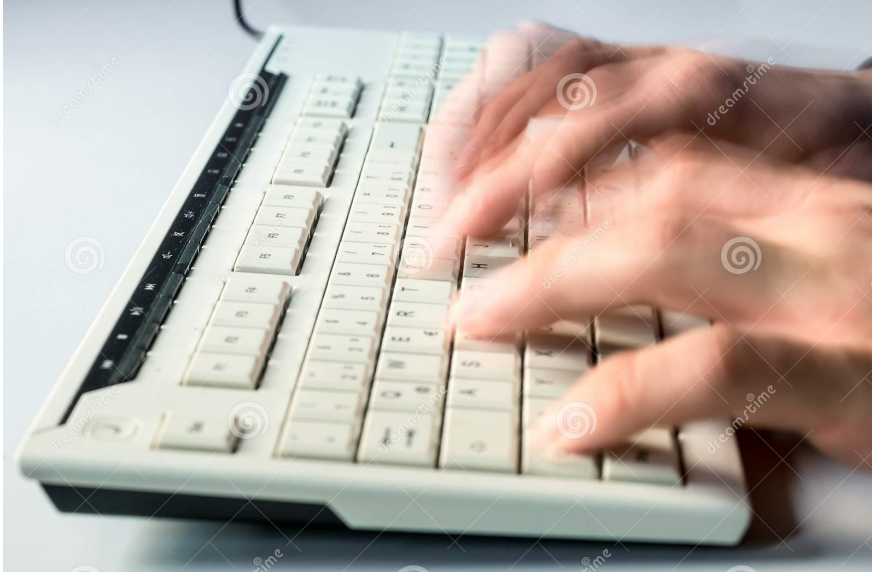
```
distance1 = math.sqrt( (x1-x2)*(x1-x2) + (y1-y2)*(y1-y2) )
distance2 = math.sqrt( (x3-x4)*(x3-x4) + (y3-y4)*(y3-y4) )
if distance1 > distance2
    ...
```

avec fonction

```
def compute_distance(x1,y1,x2,y2):
    return math.sqrt( (x1-x2)*(x1-x2) + (y1-y2)*(y1-y2) )

distance1 = compute_distance(x1,y1,x2,y2)
distance2 = compute_distance(x3,y3,x4,y4)
if distance1 > distance2
    ...
```

Fast and Keyrious v2



Continuer le programme précédent en ajoutant une fonction pour sauver des highscores sur le disque et une autre fonction pour les charger.

Rappel:

```
file = open("score.txt", "wt")
file.write(buffer)
file.close()
# plus tard:
file = open("score.txt", "rt")
buf = file.read()
file.close()
```

Highscore.py - une ebauche

```
def game():
    "Lance une partie de jeu et retourne un score. Le plus gros c'est le mieux"
    return 1.0

def load_highscore():
    "Charge le meilleur score et retourne sa valeur"
    ...

def save_highscore(score)
    "Sauve le score dans le fichier"
    ...

score = game()

best = load_highscore()

if score > best:
    print( "Bravo!\n Grace a ton score de %.3f, tu as explosé le high score de %.3f" %
(score,best) )
    save_highscore(score)
```

highscore.py

```
def fake_game():
    "Lance une partie de jeu et retourne un score. Le plus gros c'est le mieux"
    return 0.5 # une valeur pour tester rapidement

def load_highscore():
    "Charge le meilleur score et retourne sa valeur"
    filename = "save.txt"
    if not os.path.isfile(filename):
        return 0
    file = open(filename, "r")
    score = float(file.read())
    file.close()
    return score

def save_highscore(score)
    "Sauve le score dans le fichier"
    file = open("save.txt", "w")
    file.write(str(score))
    file.close()

score = fake_game()

best = load_highscore()

if score > best:
    print( "Bravo!\n Grace a ton score de %.3f, tu as explosé le high score de %.3f" % (score,best))
    save_highscore(score)
```

Rappel

- Si je veux me souvenir d'un mot, d'une heure, d'une valeur, je dois la stocker dans une variable.

```
if input("donne moi ton age") < 18:  
    print("Comme tu as " + str(?) + " tu n'as pas accès à cette ressource" )
```



- L'endroit du stockage dans une variable va avoir une incidence: si je stocke l'indice de boucle avant la boucle ou dans la boucle, ce n'est pas pareil.
- Ne pas coder quelque chose d'inutile: ça complique le programme et genere de potentiel erreur. P.ex: relire le fichier du highscore qu'on vient d'ecrire pour connaitre le high score du moment.
- Principe du KISS: Keep It Simple, Stupid

Rappel

Revoir la correction en visualisant:

- boucles
- fonctions
- usages des fichiers
- concept de test rapide

Fast and Keyrious v3



Continuer le programme précédent en ajoutant un top 5 et des noms (façon flipper).

- 1) Fonctionnaliser le programme précédent, le jeu doit être dans une fonction différente retournant un score. Il doit y avoir une autre fonction “bouchon”.
- 2) Charger le top5 depuis un fichier
- 3) Si score dans les 5 meilleurs, demander le nom du joueur.
- 4) Afficher le (nouveau) top 5
- 5) (Le stocker)

Top5.py - ébauche

```
def load_top5():
    """Charge le top5
    Retourne un tableau avec des paires [score,nom]
    par exemple: [[18,"Alex"],[17,"toto"]]
    """
    ...

def find_rank(top5,score):
    """Cherche dans le top5 la place de ce nouveau score
    Retourne la place entre 0 et 4 ou 100 si pas dans le top 5
    """
    ...

score = game()

top5 = load_top5()

rank_num = find_rank(top5,score)

if idx < 5:
    ...
    ... update le top5 ...
    save_top5(score)
```

Test unitaire

test unitaire (ou « T.U. », ou « U.T. » en anglais): une procédure permettant de vérifier le bon fonctionnement d'une partie précise d'un logiciel ou d'une portion d'un programme (appelée « unité » ou « module »).

Dans les applications non critiques, l'écriture des tests unitaires a longtemps été considérée comme une tâche secondaire. Cependant, les méthodes Extreme programming (XP) ou Test Driven Development (TDD) ont remis les tests unitaires, appelés « tests du programmeur », au centre de l'activité de programmation.

Test unitaire

Un test unitaire doit être autonome et ne pas modifier (“abimer”) le programme qu’il teste.
ou son environnement

On utilise souvent la méthode `assert(condition)` permet de vérifier une “affirmation”.
Si elle n’est pas validée, le programme s’arrete.

Essayer:

- `assert(0)`
- `assert(1)`
- `assert(False)`
- `assert(2==2)`

Test unitaire: un exemple

Partir du test unitaire peut être une bonne approche pour commencer un programme: on définit alors noir sur blanc le contrat.

```
def run_unit_test():  
    assert(multiply(2,2)==4)  
    assert(multiply(3,9)==multiply(9,3))  
    assert(multiply(13,0)==0)  
  
# ensuite je programme ma méthode:  
def multiply(a,b):  
    ...
```

Test unitaire: un autre exemple


Idée de test par rapport à notre programme “Keyrious”

```
Def run_unit_test():  
    save_highscore(300)  
    score = load_highscore()  
    assert( score == 300 )  
  
def load_highscore():  
    "Charge le meilleur score et retourne sa valeur"  
    file = open("score.txt", "rt")  
    ...  
  
def save_highscore(score)  
    "Sauve le score dans le fichier"  
    file = open("score.txt", "wt")  
    ...
```

Test unitaire: un autre exemple

Idée de test par rapport à notre programme “Keyrious”

```
Def run_unit_test():  
    save_highscore(300)  
    score = load_highscore()  
    assert( score == 300 )  
  
def load_highscore():  
    "Charge le meilleur score et retourne sa valeur"  
    file = open("score.txt", "rt")  
    ...  
  
def save_highscore(score)  
    "Sauve le score dans le fichier"  
    file = open("score.txt", "wt")  
    ...
```



Mince, j'ai pété les
vrais scores du jeu !

Test unitaire: un autre exemple corrigé

Amélioration du programme de test pour ne pas casser les scores.

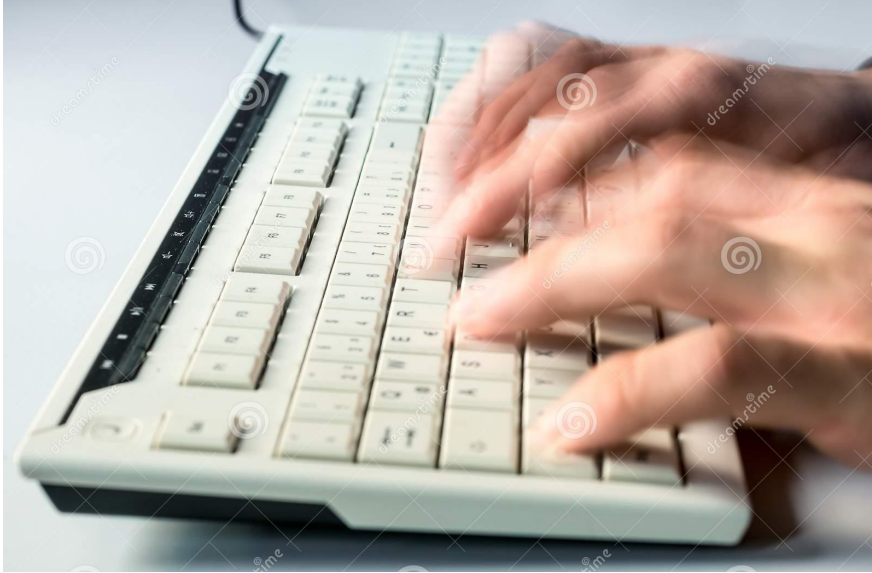
```
Def run_unit_test():
    save_highscore("test.txt", 300)
    score = load_highscore("test.txt")
    assert( score == 300 )

def load_highscore(filename_score):
    "Charge le meilleur score et retourne sa valeur"
    file = open(filename_score, "rt")
    ...

def save_highscore(filename_score, score)
    "Sauve le score dans le fichier"
    file = open(filename_score, "wt")
    ...

# plus tard dans le vrai jeu:
save_highscore("score.txt", vrai_score)
```


Fast and Keyrious v3



Continuer le programme précédent, pour rappel, on veut ajouter un top 5 et des noms (façon flipper).

- 1) Fonctionnaliser le programme précédent, le jeu doit être dans une fonction différente retournant un score. Il doit y avoir une autre fonction “bouchon”.
- 2) Charger le top5 depuis un fichier
- 3) Si score dans les 5 meilleurs, demander le nom du joueur.
- 4) Afficher le (nouveau) top 5
- 5) (Le stocker)

keyrious.py 1/2

```
def load_top5(filename):
    """Charge le top5
    Retourne un tableau avec des paires score,nom
    par exemple: [[18,"Alex"],[17,"toto"]]
    """
    if not os.path.isfile(filename):
        return []
    file = open(filename,"r")
    buf = file.read()
    print("INF: load_top5 read buffer: '" + buf
    )
    top5 = eval(buf)
    file.close()
    return top5

def save_top5(filename, top5):
    """Sauve le top5
    """
    file = open(filename,"w")
    out = "["
    for score, name in top5:
        out += "[" + str(score) + "," + name +
    ""'],"
        out += "]"
    file.write(out)
    file.close()
```

```
def find_rank(top5,new_score):
    """Cherche dans le top5 la place descore
    Retourne la place entre 0 et 4 ou 100 si pas dans le top5
    """
    i = 0
    while i < len(top5):
        score,name = top5[i]
        if new_score > score:
            print("DBG: find_rank: found better at rank: " +
str(i))
            return i
        i += 1

    if len(top5)<5:
        return len(top5)

    return 100

def update_rank(top5,new_score,new_name):
    position = find_rank(top5,new_score)
    new_top5 = top5[:4]
    new_top5.insert(position,[new_score,new_name])
    print("DBG: update_rank, top5 is now:" + str(new_top5)
    )
    return new_top5
```

keyrious.py 2/2

```
def run_unit_test():
    filename = "test.txt"
    top5 = load_top5(filename)
    assert(top5 == [])
    r = find_rank(top5, 10)
    print(r)
    assert(r == 0)
    top5 = update_rank(top5, 10, "Alex")
    assert(find_rank(top5, 10) == 1)
    assert(find_rank(top5, 12) == 0)
    top5 = update_rank(top5, 13, "Toto")
    top5 = update_rank(top5, 3, "John")
    top5 = update_rank(top5, 6, "Pat")
    top5 = update_rank(top5, 8, "Greg")
    top5 = update_rank(top5, 12, "Gg")

    assert([[13, 'Toto'], [12, 'Gg']], [10,
    'Alex'], [8, 'Greg'], [6, 'Pat']])

    assert(find_rank(top5, 10) == 3)
    assert(find_rank(top5, 5) == 100)

    save_top5(filename, top5)
    loaded_top5 = load_top5(filename)
    assert(top5 == loaded_top5)
```

```
def run_game():
    filename = "score.txt"
    score = game()

    top5 = load_top5(filename)

    rank_num = find_rank(top5, score)

    if idx < 5:
        name = input("bravo, tu es dans le top
5, entre ton nom")
        top5 = update_rank(top5, score, name)
        save_top5(filename, top5)

run_unit_test()

run_game()
```

Des questions ?

???

Tu peux me télécharger ici:



http://engrenage.studio/oia/OIA_2021_2022_Cycle1.pdf

En bonus: qrcode.py

```
import qrcode # pip3 install qrcode

data = "http://engrenage.studio/oia/OIA_2021_2022_Cycle1.pdf"

filename = "cycle1.png"

img = qrcode.make(data)

img.save(filename)
print("SUCCESS: QRCode written to file " + filename )

# et si on l'affichait juste pour voir, ca serait plus poli

import cv2

img = cv2.imread(filename)

cv2.imshow("result: " + filename, img)

cv2.waitKey(0)
```