Compléments sur les listes, tuples, dictionnaires

1. Compléments sur les listes.

Listes en compréhension

Voici un programme qui ajoute des éléments dans une liste façon successive:

In [34]:

```
L=[]
for i in range(20):
    if i%2==0:
        L.append(i)
print(L)
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

Il existe une manière plus concise de procéder, la définition en compréhension :

In [46]:

```
# Une liste en compréhension
L=[i for i in range(20) if i%2==0]
print(L)
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

Syntaxes

Voici des types d'instructions possibles pour générer des listes en compréhension:

- [*fonction de x* **for** *x in ensemble*]
- [*fonction de x* **if** *condition sur x* **else** *autre fonction de x* **for** *x in ensemble*]
- [*fonction de x* **for** *x in ensemble* **if** *condition sur x*]

Exemples:

In [14]:

```
#1
L=[3*i for i in range(11)]
print(L)

#2
from random import *
n=10
Des=[randint(1,6) for i in range(n)]
print(Des)

#3
pileface=[choice(['P','F']) for i in range(20)]
print(pileface)
```

```
[0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30]
[3, 5, 2, 5, 6, 6, 4, 5, 2, 1]
['F', 'P', 'P', 'P', 'P', 'P', 'F', 'F', 'P', 'F', 'F', 'P', 'F', 'P', 'F',
'F', 'P', 'F', 'P', 'P']
```

Exercice 1:

En utilisant la syntaxe des listes en compréhension :

1. Générer les entiers naturels de 1 à 100.
2. Générer les multiples de 5 inférieurs ou égaux à 100.
3. Générer une liste des entiers naturels de 1 à 100 dans laquelle les multiples de 5 seront remplacées par le caractère `*`

In [1]:

```
#1.
```

In [2]:

```
#2.
```

In [3]:

```
#3.
```

Exercice 2:

En utilisant les fonctions `randint()` ou `choice()` du module `random` et la syntaxe des listes en compréhension:

1. Générer une liste de 20 entiers naturels aléatoires entre 0 et 255.
2. Générer 100 caractères au hasard parmi `a` , `b` , `c`

In [4]:

```
#1.
from random import *
```

In [5]:

```
#2.
```

Listes de listes

Pour représenter des tableaux à double entrée(images, matrices...), on peut utiliser une liste de listes. On identifie ainsi un élément du tableau à l'aide de deux indexs.

Exemple :

In [31]:

```
tableau=[['A','b','c','d'],['E','f','g'],['I','j','k','m'],['N','o','p','q']]
print(tableau[0][0])
```

```
A
```

Exercice 3:

1. Quel est la lettre correpondant à `tableau[1][2]` ?
2. Quelle instruction permet d'accéder à la lettre 'm' ?
3. Ajouter au tableau la ligne `['R','s','t','u']` .
4. Ajouter le caractère `h` à sa place.
5. Remplacer la caractère `N` par `n` .

In [6]:

```
#1.
```

In [7]:

```
#2.
```

In [8]:

```
#3
```

In [9]:

```
#4
```

In [10]:

```
#5
```

Exercice 4:

Générer en comprehension une liste de 10 listes contenant chacune 10 entiers binaires aléatoires(0 ou 1).

In [ ]:

```
```

Avant de poursuivre

Nous avons étudié jusqu'ici deux types de données construits(composites) : les chaînes, qui sont composées de caractères, et les listes, qui sont composées d'éléments de n'importe quel type.

Rappel sur une différence importante entre chaînes et listes : il n'est pas possible de changer les caractères au sein d'une chaîne existante, alors que l'on peut modifier les éléments d'une liste. En d'autres termes, les listes sont des séquences modifiables, alors que les chaînes de caractères sont des séquences non-modifiables.

Exemples :

In [4]:

```
# Les listes sont modifiables
L=['a','b','d']
L[2]='c'
print(L)
```

```
['a', 'b', 'c']
```

In [3]:

```
# les chaînes ne sont pas modifiables
mot='abd'
mot[2]='c'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-3-23e01673f992> in <module>
      1 # les chaînes ne sont pas modifiables
      2 mot='abd'
----> 3 mot[2]='c'

TypeError: 'str' object does not support item assignment
```

Dans la suite de cette feuille deux nouveaux types de données : les tuples et les dictionnaires

2. tuples (ou p-uplets)

Python propose un type de données appelé tuple, qui est assez semblable à une liste mais qui, comme les chaînes, n'est pas modifiable. Du point de vue de la syntaxe, un tuple est une collection d'éléments séparés par des virgules :

In [1]:

```
#Exécuter le code ci-dessous
tup1 = 1,2,3
tup2 = (6,7)
tup3 = 'abc','def'
tup4 = ('a',1,'b',2)
print(tup1,type(tup1))
print(tup2,type(tup2))
print(tup3,type(tup3))
print(tup4,type(tup4))
```

```
(1, 2, 3) <class 'tuple'>
(6, 7) <class 'tuple'>
('abc', 'def') <class 'tuple'>
('a', 1, 'b', 2) <class 'tuple'>
```

A retenir :

- De simples virgules suffisent à définir un tuple mais pour la lisibilité du code, il est préférable de l'enfermer dans des parenthèses.

Opérations sur les tuples

In [1]:

```
#affectations
t= 7,8,9
a,b,c = t
print(a)
```

```
7
```

In [12]:

```
# opérateurs + et * : concanténation et répétition
t1=  (3,2,1)
t2 = (6,5,4)
t3 = 2*t1+t2
print(t3)
```

```
(3, 2, 1, 3, 2, 1, 6, 5, 4)
```

In [17]:

```
# Accéder aux éléments
t4 = (2,4)
print(t4[0])
```

```
2
```

In [13]:

```
# longueur d'un tuple
print(len(t3))
```

```
9
```

In [20]:

```
# Parcours d'un tuple
t5 = (0,1,2,3,4,5,6,7,8,9)
for e in t5 :
    print(e+1, end=' ')
```

```
1 2 3 4 5 6 7 8 9 10
```

In [2]:

```
# test in
b = 3 in (2,4,6)
print(b)
```

```
False
```

In [19]:

```
# les tuples ne sont pas des listes
t6 = (2,4,6,8)
t6.append(10)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-19-366c3bc37ff0> in <module>
      1 # les tuples ne sont pas des listes
      2 t6 = (2,4,6,8)
----> 3 t6.append(10)

AttributeError: 'tuple' object has no attribute 'append'
```

In [14]:

```
#Ajouter un élément à un tuple
t7 = 1,2,3,4,5
t7 = t7 + (6,) # ne pas oublier la virgule et les parenthèses
print(t7)
```

```
(1, 2, 3, 4, 5, 6)
```

A retenir :

- Les opérateurs de concaténation `+` et de multiplication `*` donnent les mêmes résultats que pour les chaînes et les listes.
- On accède aux éléments d'un tuple comme avec les chaînes et les listes.
- On peut déterminer la taille d'un tuple à l'aide de la fonction `len()` .
- On peut le parcourir à l'aide d'une boucle `for` , utiliser l'instruction `in` pour savoir si un élément donné en fait partie, exactement comme pour une liste ou pour une chaîne.
- Les tuples ne sont pas modifiables et on ne peut pas utiliser avec eux ni la méthode `.append()` ni l'instruction `del()` .
- Les tuples sont moins souples que les listes mais c'est aussi leur force. On est sûr que leur contenu ne sera pas modifié par erreur. De plus ils sont beaucoup moins gourmands en ressources et sont éxécutés plus rapidement.

Exercice 5 :

Ecrire la fonction `reverse(t)` qui prend en paramètre un tuple de trois valeurs et qui renvoie un tuple contenant les 3 valeurs dans l'ordre inverse.

Ainsi `reverse(1,2,3)` renvoie `(3,2,1)`

In [11]:

```
#Réponse
```

Exercice 6 :

Ecrire la fonction `initiales` qui prend en paramètre une chaîne de caractères de type `'NOM Prénom'` et qui renvoie un tuple contenant les initiales des noms et prénoms passés en argument.

Ainsi `initiales('John Doe')` doit renvoyer `('J','D')` .

In [12]:

```
#Réponse:
```

Exercice 7 :

En utilisant le résultat précédent, compléter la fonction `initiale_nom(noms,lettre)` qui prend en paramètres un tuple de noms et prénoms formatés comme précédemment ainsi qu'un caractère et qui renvoie un tuple contenant les chaînes avec la même initiale de nom.

Ainsi avec le tuple ci-dessous, `initiale_nom(stars,'S')` doit renvoyer `('Claude Shannon','Ivan Sutherland', 'Otto Scmitt')`

In [13]:

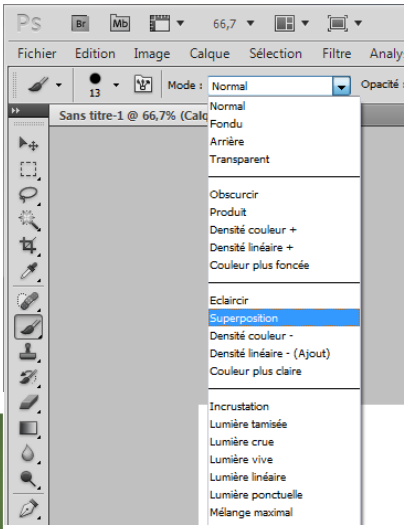
```
#Réponse
stars = ('Ada Lovelace','Alan Turing','Claude Shannon','Donald Knuth','Georges Boole','Grace Hopper',
         'Ivan Sutherland','John Von Neumann','Linus Thorvalds','Otto Schmitt','Tim Berners-Lee','Tommy Flowers')

def initiale_nom(noms, lettre):
    resultats=()




    return resultats

#Appels
initiale_nom(stars,'S')
```

Out[13]:

```
()
```

Exercice 8 :

On se place dans le contexte du codage des couleurs selon le système RGB, déjà vu dans une feuille précédente. Dans une image dite bitmap, chaque pixel contient une couleur composée de ses couches Rouge, Verte et Bleue, chacune de ces couches étant représentée par un entier compris entre 0 et 255



Dans les logiciels de traitement d'images(Photoshop, Gimp,...), on travaille avec des calques d'images que l'on superpose et que l'on fusionne. En fonction des opérations mathématiques utilisées pour fusionner les calques, on obtient des rendus esthétiques différents:

Chacune des fonctions demandées

correspondent à un mode de fusion de Photoshop , prennent en paramètres deux tuples de trois valeurs `pix1` et `pix2` correspondants aux deux pixels que l'on souhaite fusionner et renvoient un tuple contenant la couleur finale obtenue.

Pour s'aider, voici une description des formules de certains modes de fusion(en milieu de page) : https://helpx.adobe.com/fr/after-effects/using/blending-modes-layer-styles.html (https://helpx.adobe.com/fr/after-effects/using/blending-modes-layer-styles.html)

On pourra tester avec les deux pixels `p1=(200,128,63)` et `p2=(125,205,50)` . Fonctions `min()` et `max()` autorisées !

In [15]:

```
p1 = (200,128,63)
p2 = (125,205,50)
```

1. `eclaircir(p1,p2)` renvoie `(200,205,63)`

In [16]:

```
#1 : remplacer "to do" par les bonnes valeurs
def eclaircir(pix1, pix2):
    r= "to do"
    g= "to do"
    b= "to do"

    return (r,g,b)

eclaircir(p1,p2)
```

Out[16]:

```
('to do', 'to do', 'to do')
```

1. obscurcir(p1,p2) renvoie (125,128,50)

In [17]:

```
#2 :
def obscurcir(pix1, pix2):
    r= "to do"
    g= "to do"
    b= "to do"

    return (r,g,b)

obscurcir(p1,p2)
```

Out[17]:

```
('to do', 'to do', 'to do')
```

1. difference(p1,p2) renvoie (75,155,13)

In [18]:

```
#3 :
def difference(pix1,pix2):
    r= "to do"
    g= "to do"
    b= "to do"

    return (r,g,b)

difference(p1,p2)
```

Out[18]:

```
('to do', 'to do', 'to do')
```

1. addition(p1,p2) renvoie (255,255,113)

In [19]:

```
#4 :
def addition(pix1,pix2):
    r= "to do"
    g= "to do"
    b= "to do"

    return (r,g,b)

addition(p1,p2)
```

Out[19]:

```
('to do', 'to do', 'to do')
```

1. soustraction(p1,p2) renvoie (0,77,0)

In [20]:

```
#5 :
def soustraction(pix1,pix2):
    r= "to do"
    g= "to do"
    b= "to do"

    return (r,g,b)

soustraction(p1,p2)
```

Out[20]:

```
('to do', 'to do', 'to do')
```

1. produit(p1,p2) renvoie (98,103,12)

In [21]:

```
#6 :
def produit(pix1,pix2):
    r= "to do"
    g= "to do"
    b= "to do"

    return (r,g,b)

produit(p1,p2)
```

Out[21]:

```
('to do', 'to do', 'to do')
```

1. division(p1,p2) renvoie (159,255,202)

In [22]:

```
#7 :
def division(pix1,pix2):
    r= "to do"
    g= "to do"
    b= "to do"

    return (r,g,b)

division(p1,p2)
```

Out[22]:

```
('to do', 'to do', 'to do')
```

1. superposition(p1,p2) renvoie (226,230,100)

In [23]:

```
#8 :
def superposition(pix1,pix2):
    r= "to do"
    g= "to do"
    b= "to do"

    return (r,g,b)

superposition(p1,p2)
```

Out[23]:

```
('to do', 'to do', 'to do')
```

3. Dictionnaires

Définition et création

Les types de données construits que nous avons abordés jusqu'à présent (chaînes, listes et tuples) sont tous des séquences, c'est-à-dire des suites ordonnées d'éléments. Dans une séquence, il est facile d'accéder à un élément quelconque à l'aide d'un index (un nombre entier), mais encore faut-il le connaître.

Les dictionnaires constituent un autre type construit. Ils ressemblent aux listes(ils sont modifiables comme elles), mais ce ne sont pas des séquences. Les éléments que nous allons y enregistrer ne seront pas disposés dans un ordre immuable. En revanche, nous pourrons accéder à n'importe lequel d'entre eux à l'aide d'un index que l'on appellera une clé, laquelle pourra être alphabétique ou numérique.

Comme dans une liste, les éléments mémorisés dans un dictionnaire peuvent être de n'importe quel type(valeurs numériques, chaînes, listes ou encore des dictionnaires, et même aussi des fonctions).

Exemple :

In [7]:

```
dico1={}
dico1['nom']='Paris-Orly Airport'
dico1['ville']='Paris'
dico1['pays']='France'
dico1['code']='ORY'
dico1['gps']=(48.7233333,2.3794444)

print(dico1)
print(dico1['code'])
print(dico1['gps'])
```

```
{'nom': 'Paris-Orly Airport', 'ville': 'Paris', 'pays': 'France', 'code':
'ORY', 'gps': (48.7233333, 2.3794444)}
ORY
(48.7233333, 2.3794444)
```

Remarques :

- Des accolades délimitent un dictionnaire.
- Les éléments d'un dictionnaire sont séparés par une virgule.
- Chacun des éléments est une paire d'objets séparés par deux points : une clé et une valeur.
- La valeur de la clé `'code'` est `'ORY'`.
- La valeur de la clé `'gps'` est un tuple de deux flottants.

Exercice 9 :

Voici des données concernant l'aéroport international de Los Angeles :

- **"Los Angeles International Airport","Los Angeles","United States","LAX",33.94250107,-118.4079971**

En utilisant les mêmes clés que dans l'exemple précédent, créer le dictionnaire 'dico2' qui contiendra les données ci-dessus

In [24]:

```
#Réponse
```

Méthodes

In [41]:

```
#Afficher les clés
print(dico1.keys())

#Affichées les valeurs
print(dico1.values())

#Afficher les éléments
print(dico1.items())
```

```
dict_keys(['nom', 'ville', 'pays', 'code', 'gps'])
dict_values(['Paris-Orly Airport', 'Paris', 'France', 'ORY', (48.7233333,
2.3794444)])
dict_items([('nom', 'Paris-Orly Airport'), ('ville', 'Paris'), ('pays', 'Fr
ance'), ('code', 'ORY'), ('gps', (48.7233333, 2.3794444))])
```

Parcours

On peut traiter par un parcours les éléments contenus dans un dictionnaire, mais attention :

- Au cours de l'itération, **ce sont les clés** qui servent d'index
- L'ordre dans lequel le parcours s'effectue est imprévisible

Exemple :

In [8]:

```
for element in dico1:
    print(element)
```

```
nom
ville
pays
code
gps
```

Exercice 10 :

Modifier le programme ci-dessus pour obtenir l'affichage ci-dessous :

In [ ]:

```

```

4. Exercices

Exercice 11 :

Lors d'une élection, entre 2 et 6 candidats se présentent. Il y a 100 votants, chacun glisse un bulletin avec le nom d'un candidat dans l'urne. Les lignes de code ci-dessous simulent cette expérience.

In [26]:

```
from random import *

#liste des candidats potentiels
noms=['Alice','Bob','Charlie','Daniella','Eva','Fred']

#liste des candidats réels
candidats=list(set([choice(noms) for i in range(randint(2,len(noms)))]))

#nombre de votants
votants=100

#liste des bulletins dans l'urne
urne=[choice(candidats) for i in range(votants)]

print('Candidats : ',candidats)
#print("Contenu de l'urne :", urne)
```

```
Candidats :  ['Eva', 'Charlie']
```

1. Vérifier que les candidats réels changent à chaque éxecution de la cellule ci-desssus ainsi que le contenu de l'urne.

1. Compléter la fonction `depouillement(urne)` . Elle prend en paramètre une liste (la liste des bulletins exprimés) et renvoie un dctionnaire. Les paires clés-valeurs sont respectivement constituées du noms d'un candidat réels et du nombre de bulletins exprimés en sa faveur. Par exemple, si la liste des candidats est `['Alice','Charlie,'Bob']` , le résultat pourra s'afficher sous la forme `{'Eva': 317, 'Charlie': 363, 'Alice': 320}`

In [27]:

```
# Remplacer "pass" par les bonnes instructions
def depouillement(urne):
    decompte={}

    for bulletin in urne:
        if bulletin in decompte:
            pass
        else:
            pass

    return decompte


depouillement(urne)
```

Out[27]:

```
{}
```

1. Ecrire la fonction `vainqueur(election)` qui prend en paramètre un dictionnaire contenant le décompte d'une urne renvoyé par la fonction précédente et qui renvoie le nom du vainqueur.

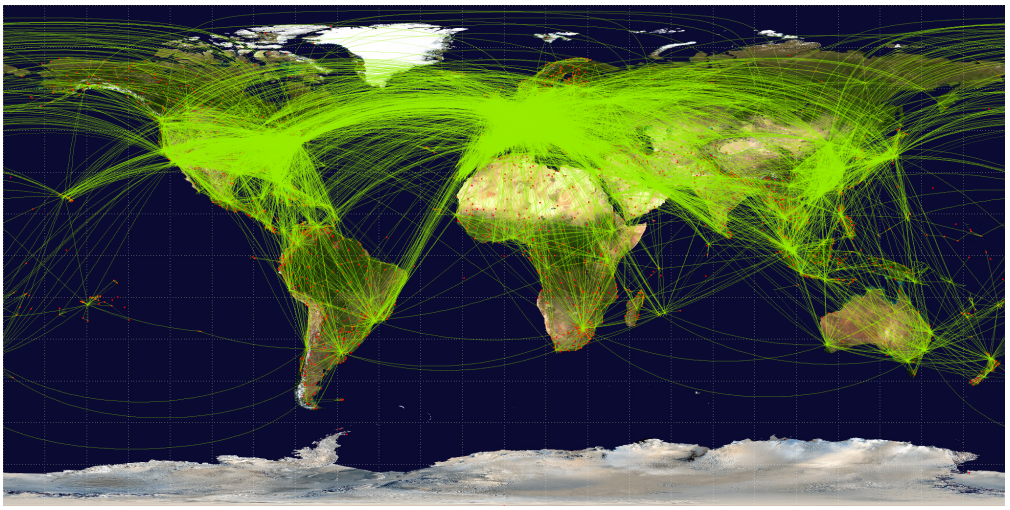
In [32]:

```
def vainqueur(election):
    vainqueur=''




vainqueur(depouillement(urne))
```

Out[32]:

```
'Bob'
```

Exercice 12



Sur le site https://openflights.org/data.html (https://openflights.org/data.html) , on trouve des bases de données mondiales aéronautiques.Le fichier `airports.txt` présent dans le dossier de cette feuille contient des informations sur les aéroports.

Chaque ligne de ce fichier est formatée comme l'exemple ci-dessous :

```
1989,"Bora Bora Airport","Bora Bora","French Polynesia","BOB","NTTB",-16.444400787353516,-151.75100708007812,10,-10,"U","Pacific/Tahit
```

On souhaite extraire les informations suivantes pour chaque aéroport:

- Sa référence unique, un entier.
- Le nom de l'aéroport, une chaîne de caractères
- La ville principale qu'il dessert, une chaîne de caractères
- Le pays de cette ville,une chaîne de caractères
- Le code IATA de l'aéroport composé de 3 lettres en majuscules
- Ses coordonées gps (latitude puis longitude), un tuple de 2 flottants.

1. Compléter les champs ci-dessous pour l'aéroport cité en exemple:

- ref :
- nom :
- ville :
- pays :
- gps :

2. La fonction `data_extract` doit parcourir le fichier et extraire les données demandées qu'elle renvoie sous forme d'une liste de dictionnaires.

- Chaque élément de la liste est donc un dictionnaire qui correspond à un aéroport.
- Les clés sont les noms des champs que l'on souhaite extraire et les valeurs sont celles associées à chaque aéroport.

Recopier , modifier et compléter cette fonction pour qu'elle éxécute la tâche demandée :

In [32]:

```
#2.
def data_extract(chemin):
    fichier = open(chemin, "r",encoding='utf-8')
    res = [] # Pour contenir le résultat qui sera une liste de dictionnaires
    for ligne in fichier:
        datas = ligne.strip().split(",")  # une ligne du fichier
        res.append(
        { "ref": int(datas[0]),
          "nom": datas[1][1:-1],
          "ville": "A compléter",
          "pays": datas[3][1:-1],
          "A compléter": datas[4][1:-1],
          "gps" : "A compléter"
        })

    fichier.close()
    return res

airports=data_extract('airports.txt')

#nombre d'aéroports référencés
print("A compléter")

#un aéroport au hasard
print(choice(airports))
```

```
A compléter
{'ref': 8631, 'nom': 'Capital City Airport', 'ville': 'A compléter', 'pay
s': 'United States', 'A compléter': 'FFT', 'gps': 'A compléter'}
```

In [28]:

```
#2. Réponse
```

3. A l'aide d'une liste en compréhension, récupérer la liste des villes françaises desservies par un aéroport

In [30]:

```
#3. Réponse
country='France'
res=['A compléter']
```

4. Ecrire la fonction `infos(airports,ville)` .

Elle prend en paramètres la liste des dictionnaires de données extraites et une chaîne de caractères(le nom d'une ville). Elle renvoie les informations du ou des aéroports de cette ville dans une liste de dictionnaires:

In [33]:

```
#4.
def infos(airports,ville):
    res=[]




    return res

print(infos(airports,'Nice'))
```

```
[]
```

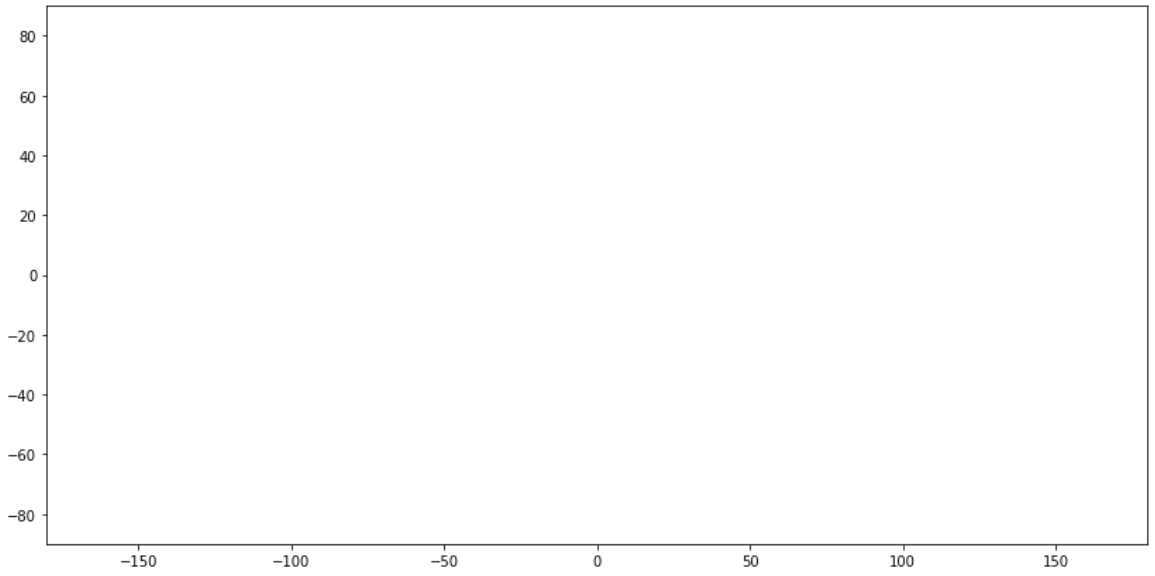
5. Compléter les listes du code ci-dessous pour représenter les points de coordonnées gps de chacun des aéroports de la base de données(liste `X` des longitudes et liste `Y` des latitudes)

In [131]:

```
#5.
from matplotlib import pyplot #bibliothèque pour tracer des graphiques

X=[] #liste des longitudes
Y=[] #liste des latitudes

pyplot.figure(figsize = (14,7))
pyplot.xlim(-180, 180)
pyplot.ylim(-90, 90)

pyplot.plot(X,Y,'rx') # g=green, x est une croix
```

Out[131]:

```
[<matplotlib.lines.Line2D at 0xef48b0>]
```



6. Recopier et modifier le code précédent pour faire apparaître en rouge les aéroports situés en zone tropicale (c'est à dire dont la latitude est comprise entre -23 et $+23$)et en bleu les autres

In [34]:

```
#5.Réponse
```

FIN