# Les bases de python

Meilland Jean claude

August 30, 2017

## 1 Hello World

On ne va pas y échapper, quand on apprend un langage, le premier programme étudié est le programme qui affiche "Hello World!". On utilisera ipython qui est shell python amélioré.

```
In [ ]: print("Hello World !")
```

C'est relativement simple.

Remarquez au passage l'absence de point-virgule en fin de ligne, contrairement au C, à Java et à bon nombre de langages.

## 2 La syntaxe générale

### 2.1 indentation

Python oblige donc le développeur à structurer son code a l'aide des ***indentations*** : ce sont elles qui détermineront les blocs (séquences d'instructions liées) et non les accolades comme dans la majorité des langages.

### 2.2 Commentaires

Le caractère **#** permet d'indiquer que tous les caractères suivants sur la même ligne ne doivent pas être interprétés. il s'agit de commentaires

### 2.3 Variables

Python utilise le symbole **=** pour affecter une valeur à une variable.
par exemple:

```
In [ ]: a=2
```

```
In [ ]: a==12
```

Attention: a=12 place la valeur 12 dans la variable a mais a==12 vérifie si la variable a est égale à 12, c'est donc un boléen. (***True*** ici car a est bien égal à 12).

```
In [ ]: a=2
```

```
In [ ]: a

In [ ]: a+=3

In [ ]: a
```

Les opérateurs d'incrémentation ou de décrémentation ++ ou - n'existent pas en Python.
Par contre, vous avez la possibilité d'effectuer des affectations de variables multiples avec une même valeur :

```
In [ ]: a=b=1

In [ ]: a

In [ ]: b
```

Et vous pouvez également effectuer des affectations multiples de variables ayant des valeurs différentes en séparant les noms de variables et les valeurs par des virgules :

```
In [ ]:  a,b = 2,3

In [ ]: a

In [ ]: b
```

Ce mécanisme peut paraître anodin, mais il permet de réaliser très simplement une permutation de valeurs sans avoir recours a une variable intermédiaire :

```
In [ ]: a=2
        b=3
        a,b = b,a

In [ ]: a

In [ ]: b
```

### 2.4 Les opérateurs

Il existe de nombreux opérateurs comme l'addition **+** la soustraction **-** la multiplication ***** et la division \

En voici d'autre que l'on utilise souvent.

7**//**2 est le ***quotient*** de la division euclidienne de 7 par 2

```
In [ ]: 7//2
```

7%2 est le ***reste*** de la division euclidienne.

```
In [ ]: 7%2
```

7**2 pour 7 à la ***puissance*** 2

```
In [ ]: 7**2
```

***sqrt(2)*** pour la ***racine carré*** de 2.
Attention il faut auparavant importer la librairie ***math***

```
In [ ]: import math
        math.sqrt(2)
```

ou alors

```
In [ ]: from math import sqrt
        sqrt(2)
```

# 3 Les différents types de données

Pour pouvoir travailler sur les types, nous aurons besoin de faire appel à une fonction : la fonction ***type()***.
Le rôle de cette fonction est d'afficher le type de l'élément qui lui est passé en paramètre.

## 3.1 Les données numériques

Parmi les données manipulables, on va bien entendu pouvoir utiliser des nombres. Ceux-ci sont classés en quatre types numériques :
### Entier
Python permet de représenter les entiers sur 32 bits (de -2147483648=-$2^{31}$ à 2147483647=$2^{31}-1$).
La représentation des entiers dépend de la version python 2 ou 3, du système utilisé (32 ou 64 bit).
On utilisera le module sys pour connaitre la valeur max.

```
In [ ]: type(2)
```

```
In [ ]: import sys
        sys.maxsize
```

```
In [ ]: type(sys.maxsize)
```

```
In [ ]: type(sys.maxsize+1)
```

Des préfixes particuliers permettent d'utiliser la notation octale (en base huit) ou hexadécimale (en base seize). - Pour la notation octale, il faudra préfixer les entiers par ***0o*** (le chiffre zéro suivi de la lettre << o >> en minuscule).
Par exemple, 12 en notation octale vaut 8+2=10 :

```
In [ ]: 0o12
```

- Pour la notation hexadécimale, le préfixe sera 0x (le chiffre zéro suivi de la lettre << x >> en minuscule).
  Ainsi, 12 en notation hexadécimale vaut 16+2=18 :

```
In [ ]: 0x12
```

On peut convertir aussi un nombre ou un texte en entier avec la fonction ***int()***

```
In [ ]: type(int("16"))
```

### 3.1.1 Flottant

pour les nombres flottants. Le symbole utilisé pour déterminer la partie décimale est le point. On peut également utiliser la lettre E ou e pour signaler un exposant en base 10 (101, 102, etc.).

type(3.14)

```
In [ ]: 2e3

In [ ]: type(2e1)
```

On peut convertir aussi un nombre ou un texte en flottant avec la fonction ***float()***

```
In [ ]: float(3)

In [ ]: type(float(3))

In [ ]: float('3')
```

### 3.1.2 Complexe

Pour les nombres complexes qui sont représentés par leur partie réelle et leur partie imaginaire. En Python, la partie imaginaire sera logiquement suivie de la lettre J ou j.
En informatique, on utilise souvent la variable i comme variable de boucle (dont nous parlerons par la suite), Pour éviter toute confusion, la partie imaginaire des complexes se note donc j. Voici comment écrire le complexe 2 + 3i :

```
In [ ]: 2+3j

In [ ]: type(2+3j)
```

### 3.1.3 Problème avec les flottants

L'interpréteur interactif peut être utilisé comme une calculette. Pour l'instant nous ne parlerons que des opérations élémentaires **+,-,*** et **/** auxquelles nous ajouterons le quotient de la division euclidienne noté **//** (ou ***div*** ) et le reste de la division euclidienne noté **%** (ou ***mod*** ). Voici quelques exemples de calculs :

```
In [ ]: 7/3

In [ ]: 7.0/3

In [ ]: 7//3

In [ ]: 7%3
```

Les résultats paraissent ici logiques: 7/3 et 7.0/3.0 donnent le même résultat sous la forme d'un flottant et 7//3 a pour résultat 2 qui est bien le quotient de la division euclidienne de 7 par 3. Si vous utilisez les calculs en Python, l'usage de la division peut donc être source d'erreur et il faudra donc bien se souvenir de la version de Python employée. Mais il y a pire ... et ce n'est pas l'apanage de Python.

Le phénomène que je m'apprête à souligner est vrai pour absolument tous les langages, mais il est bien souvent oublié. Quel est le résultat d'une opération simple telle que 0.7 + 0.1 ?

Vous pensez que le résultat est 0.8 ? Testons donc en Python :

```
In [ ]: 0.7+0.1
```

On dirait qu'il y a un problème ... En fait cela provient de la représentation des flottants en machine :
ils sont arrondis ! En Python, la solution consistera a utiliser un module spécial, le module ***Decimal***.

```
In [ ]: from decimal import *
        Decimal('0.1')+Decimal('0.7')
```

**Si vous manipulez des flottants pour des calculs précis, méfiez vous !**

### 3.2 Les chaîne caractères

Les chaînes de caractères sont encadrées par des apostrophes ou des guillemets. Le choix de l'un ou de l'autre n'a aucun impact au niveau des performances.
La concaténation (assemblage de plusieurs chaînes pour n'en produire qu'une seule) se fait à l'aide de l'opérateur +. Enfin, l'opérateur * permet de répéter une chaîne. Voyons cela sous forme d'exemples :

```
In [ ]: a="Hello "
        b="World !"
        a+b
```

```
In [ ]: 3*a
```

L'accès à un caractère particulier de la chaîne se fait en indiquant sa position entre crochets (le premier caractère est à la position 0) :

```
In [ ]: a[0]
```

En Python, on peut faire beaucoup de choses avec les chaînes de caractères ... nous verrons cela plus en détail dans l'article consacré aux chaînes et aux listes.

### 3.3 Les boléens

Les valeurs booléennes sont notées ***True*** et ***False*** (attention à la majuscule).

Dans le cadre d'un test, les valeurs ***0***, ”” (la chaîne vide) et ***None*** sont considérées comme étant égales à ***False***. Ces valeurs sont retournées comme résultats des tests réalisés à l'aide des opérateurs de comparaison :
**==** pour l'égalité, **!=** pour la différence, puis **<**, **>**, **<=** , et **>=**.
Exemple :

```
In [ ]: 2==3
```

```
In [ ]: 2<=3
```

Pour combiner les tests, on utilise les opérateurs booléens (ou connecteurs logiques):
***and*** (et), ***or*** (ou), ***xor*** et ***not*** (non).

- et : et est vraie si les deux conditions sont vraies simultanément.
- ou : ou est vraie si au moins une des deux conditions est vraie.
- xor : xor est un ou exclusif (eXclusive OR). La condition xor est vraie si exactement une des deux conditions est vraie.
- non : non() est le contraire de .

| **et** | **V** | **F** |
|---|---|---|
| **V** | V | F |
| **F** | F | F |

| **ou** | **V** | **F** |
|---|---|---|
| **V** | V | V |
| **F** | V | F |

| **xor** | **V** | **F** |
|---|---|---|
| **V** | F | V |
| **F** | V | F |

| **not** | **V** | **F** |
|---|---|---|
| | F | V |

Voici Un comparatif entre l'algorithmique et python:

| Algorithmique | python |
|---|---|
| a=b | a==b |
| a≠b | a!=b |
| A et B | A and B A & B |
| A ou B | A or B A \| B |
| A xor B | A ^ B |
| non A | not(A) |

```
In [ ]: 1>2 and True
```

```
In [ ]: 2>1 and True
```

```
In [ ]: 1>2 or True
```

### 3.4 Les listes, les tuples et les dictionnaires

Nous allons maintenant voir trois types particuliers présentant de fortes similitudes : les listes, les tuples et les dictionnaires. Ces types étant un peu complexes, il ne s'agit ici que d'un premier aperçu, nous les étudierons plus en détail dans l'article leur étant consacré.
### Les listes Une liste est un ensemble d'éléments éventuellement de différents types. Une liste se définit à laide des crochets et une liste vide est symbolisée par [].
L'accès à un élément de la liste se fait en donnant son index, de la même manière qu'avec les chaînes de caractères :

```
In [ ]: maliste=[3.14, "cours isn", 1+1j, 5]
        maliste[1]
```

#### 3.4.1 Les tuples

Les tuples sont des listes particulières, on ne peux pas les modifier, nous verrons l'intérêt plus tard. Ils sont définis par des parenthèses et leurs éléments sont accessibles de la même manière que pour les listes.
Notez que pour lever toute ambiguïté, un tuple ne contenant qu'un seul élément sera noté (élément,). Si vous omettez la virgule, Python pensera qu'il s'agit d'un parenthèsage superflu et vous n'aurez donc pas créé un tuple.

```
In [ ]: t=(2,3)
        type(t)
```

```
In [ ]: type((2))

In [ ]: type((2,))
```

L'exemple des listes peut tout a fait être rapporté aux tuples :

```
In [ ]: montuple=(3.14, "cours isn", 1+1j, 5)
        montuple[1]
```

### 3.4.2 Les dictionnaires

Un dictionnaire est une liste où l'accès aux éléments se fait à l'aide d'une clé alphanumérique ou purement numérique. Il s'agit d'une association clé/valeur sous la forme ***clé:valeur***.
Les dictionnaires sont définis a l'aide des **accolades**. Dans d'autres langages les dictionnaires sont aussi appelés tableaux associatifs ou tables de hachage. Voici un exemple d'utilisation d'un dictionnaire :

```
In [ ]:  t ={ 'nom' : 'Meilland' , 'prenom' : 'jean claude' }
         t['nom']
```

## 3.5 Le type byte

Le type ***byte*** est un type souvent utilisé avec python3. C'est un tableau d'octets.

```
In [ ]: unbyte=b'ceci est un byte'
        print(unbyte)
        type(unbyte)
```

À quoi sert un byte?
Généralement à tout échange de données. Par exemple:
- Executer un logiciel quelconque sur l'ordinateur, - Parser une page internet (analyse syntaxique qui permet d'obtenir le code de la page internet), - lire un fichier texte - etc..
Voici un exemple:

```
In [ ]: a='Pablo Neruda'
        b=a.encode('utf8')
        print(a)
        print(b)
        print(type(a))
        print(type(b))
```

Le ***b*** est là pour signifier que la type est byte

```
In [ ]: b=a.decode('utf8')
        print(b)
        print(type(b))
```

On peut passer du type byte à string en encodant en utf8 par exemple.

### 3.6 retenir

Les différents types entiers, flottants, complexes, chaînes de caractères, boléens, listes, tuples et dictionnaires.

- ***entier***: 3
- ***flottant***: 2.3
- ***complexe***: 1+2j
- ***chaînes de caractères (string)***: 'ISN'
- ***Boléen***: True
- ***Listes***: [0,1,2,3]
- ***Tuples***: (0,1,2,3)
- ***Dictionnaires***: {'zero':0, 'un':1, 'deux':2,'trois':3}
- ***Byte***: b'toto'

On aura toujours à l'esprit que 0.2+0.7=0.8999999999999999$\neq$ 0.9

- ***int()*** permet de convertir, un nombre ou une chaîne de caractère en un entier.
- ***float()*** permet de convertir, un nombre ou une chaîne de caractère en un flottant.
- Quelques opérateur déjà vu et très utiles

| **Opération** | **algorithme** | **python** |
|---|---|---|
| addition | 2+3 | 2+3 |
| Soustraction | 12-5 | 12-5 |
| Multiplication | 3*6 | 3*6 |
| Division | 7/2 | 7/2 |
| Quotient de la division euclidienne | 7 div 2 ou div(7,2) | 7//2 |
| Reste de la division euclidienne | 7 mod 2 ou mod(7,2) | 7%2 |
| puissance | $7^2$ | 7**2 |
| racine carrée | $\sqrt{2}$ ou sqrt(2) | sqrt(2) |

## 4 Les entrées et les sorties E/S

### 4.1 Entrer des données

Il est parfois nécessaire d'entrer des données

```
In [ ]: a=input("Entrer un texte a: ")
        type(a)
```

```
In [ ]: b=int(input("Entrer un entier b: "))
        type(b)

In [ ]: c=float(input("Entrer un réel c: "))
        type(c)
```

Attention *b=int(input("Entrer un entier b: "))* indique une erreur si on saisit par exemple 2.3 Pour arrondir 2.3 à 2 On utilisera ***float*** et ***int***.

```
In [ ]: d=int(float(input("Entrer un réel c: ")))
        print(d)
```

## 4.2 Afficher un texte ou la valeur d'une variable

On utilise pour afficher un texte ou la valeur d'une variable la fonction ***print()***

```
In [ ]: print('toto')

In [ ]: a='Meilland jean claude'
        b='Lebowski Jeff'
        print(a,b)

In [ ]: print(a)

In [ ]: print(a,b,sep='\n')  # sep signifie séparateur  et \n pour newline
```

On peut assi afficher du texte et des variables. Il existe plusieurs méthodes
### méthode 1

```
In [ ]: a=12
        print("La valeur de la variable a est ",str(a),".",sep='') # par defaut sep est un espac
```

### ii. Méthode 2

```
In [ ]: print("La valeur de la variable a est " + str(a) + ".") # autre forme avec des +
```

### 4.2.1 Méthode 3 avec la méthode format()

Avec python3 il est conseillé d'utiliser cette méthode

```
In [ ]: print("La valeur de la variable a est {}.".format(a)) # Cette méthode est à préférer
```

On remplace {} par le contenu de la variable a. Peu importe ici si a est un texte, un flottant Le résultat sera affiché.

Voici des exemple un peu plus complexe

```
In [ ]: a=4
        b=5
        print("Le produit de {} par {} est {}".format(a,b,a*b))

In [ ]:  print("{0}*{1}={2} et {0}/{1}={3}".format(a,b,a*b,a/b))
```

### 4.3 À retenir

- On utilise le caractre **#** pour écrire un commentaire
- Python utilise le symbole **=** pour affecter une valeur à une variable.
  Attention à ne pas confondre a=12 et a==12.
- ***input("texte")*** permet de saisir du texte pour un programme.

- Il faudra éventuellement convertir ce texte dans le type voulu avec ***int()*** ou ***float()***
- On utilise de préférence la méthode ***format()*** pour afficher du texte et des variables.

  print("Le produit de {} par {} est {}".format(a,b,a*b))

## 5 Les structures de boucle et de test

### 5.1 Les structures de test

#### 5.1.1 Si ... alors ... sinon

if condition: # Traitement bloc 1 else: # Traitement bloc 2

Pour les tests multiples, il faudra enchaîner les if en cascade grâce à l'instruction elif, contraction de else if :

if condition: # Traitement bloc 1 elif: # Traitement bloc 2 else: # Traitement bloc 3

Par exemple:

```
In [ ]: a=3
        if a>1:
            print('a>1')
        else:
            print('a<=2')
```

#### 5.1.2 try (pour gérer les erreurs)

Ce n'est pas exectement unestructure de test mais elle s'en rapproche.

try: # commandes except: # traitement des erreurs finaly: # commandes à exécuter dans tous les cas

par exemple:

```
In [ ]: a=1
        b=0
        try:
            c=a/b
            print(c)
        except (ZeroDivisionError): print('Vous ne pouvez pas diviser par 0')
        except: print('il y a une autre erreur')
```

Modifier le programme précédent en modifiant la 2° ligne par:

b='a'

```
In [ ]: try:
            a=int(input("saisir un entier"))
            print("Vous avez l'entier {}".format(a))
        except:
            print("Il fallait entrer un entier")
```

Voici une variante que j'utilise de temps en temps

```
In [ ]: while 1:
            try:
                a=int(input("saisir un entier"))
            except:
                pass
            else:
                break
        print("Vous avez l'entier {}".format(a))
```

Le programme demande de saisir un entier jusqu'au moment où l'on rentre un entier.
Avec ***try*** on est obligé d'utiliser ***except***, c'est pourquoi j'utlise ***pass***.

## 5.2 Les structures de boucle

Pour les boucles il existe deux structures.

### 5.2.1 La boucle for

La boucle ***for*** prend ses valeurs dans une liste :
**for *variable* in *liste_valeurs*** . À chaque itération la variable prendra pour valeur un élément de la liste.

```
In [ ]: for e in [ 'pomme' , 'poire' , 'kiwi' ]:
            print(e)
```

Pour créer une boucle équivalente aux boucles traditionnelles << pour i allant de m à n, faire ... >>, nous utiliserons la fonction ***range()***.
La syntaxe générale est ***for i in range(m,n,p):***
**i prend alors toutes les valeurs de m à n-1 par pas de p**

```
In [ ]: for i in range(0,4):
            print(i)

In [ ]: for i in range(4):
            print(i)

In [ ]: for i in range(0,6,2):
            print(i)

In [ ]: for i in range(4,0,-1):
            print(i)
```

Le mot-clé ***continue*** permet de passer à l'itération suivante et le mot-clé ***break*** permet de sortir de la boucle :

```
In [ ]: for i in range(0,10):
            if i==2:
                continue
            if i==6:
                break
            print(i)
```

### 5.2.2 La boucle while

La seconde structure de boucle est le << *tant que* >> : tant qu'une condition est vraie, on boucle. Attention : dans ce type de boucle on utilise une variable de boucle dont la valeur doit changer pour pouvoir sortir de la boucle :

```
In [ ]: i=0
        while i<4:
            print(i)
            i+=1 #équivalent à i=i+1
```

La boucle ***while*** accepte elle aussi les mots-clés ***continue*** et ***break***.

## 5.3 À retenir.

Il faut absolument savoir utiliser les test ***if*** et boucles ***for*** et ***while***

Il n'y a que les exercices qui vous permettrons de vous familiariser avec ceci.

# 6 La structure d'un programme

Avec L'ensemble des instructions que nous avons vu, vous pouvez écrire des programmes plus longs que les quelques lignes testées dans l'interpréteur interactif. Vous allez donc avoir besoin de stocker vos scripts dans des fichiers d'extension **.py**.

Pour exécuter ces fichiers de code vous aurez alors deux possibilités : - soit lancer simplement dans un terminal :
python3 monFichier.py - soit ajouter à votre code source une ligne indiquant où se trouve l'interpréteur Python à utiliser :

```
In [2]: #!/usr/bin/python3
```

Cette ligne doit être la première ligne de votre fichier. Ensuite vous n'avez plus qu'a rendre votre fichier exécutable (chmod u+x monFichier.py ) et vous pourrez lancer votre script par : ./monFichier.py

# 7 Exercices

**Exercice 1**: Demander à l'utilisateur de saisir les longueurs et largeurs d'un rectangle, afficher sa surface.

In [ ]:

**Exercice 2**: Saisir une note, afficher "Ajourné" si la note est inférieure à 8, "Oral vous avez eu << la note >>" entre 8 et 10, "Admis" au dessus de 10.

In [ ]:

**Exercice 3**: Saisir trois entiers a,b et c. Puis déterminer le nombre de racines ainsi que les racines du polynôme $ax^2 + bx + c$

Aide! Pour utiliser la racine carré (square root en anglais) il faut importer la librairie ***math***. Par exemple pour calculer $\sqrt{(2)}$

import math math.sqrt(2)

In [ ]:

**Exercice 4**: Une compagnie d'assurance effectue des remboursements en laissant une somme, appelée franchise, à la charge du client. La franchise représente 10% du montant des dommages sans toutefois pouvoir être inférieure à 15 euros ou supérieure à 500 euros.

Écrire un algorithme demandant à l'utilisateur de saisir le montant des dommages et lui affichant le montant payé par l'assurance ainsi que le montant de la franchise.

In [ ]:

**Exercice 5**: Qu'affiche le programme suivant:

a=1
b=2
if a>=b: a=b else: b=a print("a={} et b={}".format(a,b))

In [ ]:

**Exercice 6**: Saisir une valeur et afficher sa valeur absolue (sans utiliser math.fabs bien sur). Méthode 1

Méthode2

In [ ]:

**Exercice 7**: Saisir trois valeurs puis afficher la plus petite de ces valeurs.

In [ ]:

**Exercice 8**: Saisir trois valeurs puis afficher le nombre de valeurs égales.

In [ ]:

**Exercice 9**: La factorielle de l'entier $n$ est le nombre noté $n!$ avec $n! = n \times (n-1) \times \cdots \times 2 \times 1$. Écrire un algorithme calculant la factorielle d'un nombre saisi par l'utilisateur.

In [ ]:

**Exercice 10**: Écrire un algorithme qui créer un nombre aléatoire entre 1 et 1000.
il faudra ensuite trouver ce nombre en saisissant des nombres tant que que le nombre mystère n'a pas été trouvé, et afficher à chaque fois le texte "c'est plus" (le nombre proposé est trop petit) ou "c'est moins" (le nombre proposé est trop grand) selon les cas.
À la fin il faudra afficher le texte "Nombre d'essais nécessaires : " puis, le nombre d'essais qui ont été nécessaires.

On utilisera pour générer le nombre aléatoire entre 1 et 1000 le code ci-dessous
import random mystere=random.randint(1,1000)

In [ ]:

**Exercice 11**: Écrire un algorithme qui affiche les tables de multiplications comme ci-dessous.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **1** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| **2** | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 |
| **3** | 3 | 6 | 9 | 12 | 15 | 18 | 21 | 24 | 27 |
| **4** | 4 | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 36 |
| **5** | 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 |
| **6** | 6 | 12 | 18 | 24 | 30 | 36 | 42 | 48 | 54 |
| **7** | 7 | 14 | 21 | 28 | 35 | 42 | 49 | 56 | 63 |
| **8** | 8 | 16 | 24 | 32 | 40 | 48 | 56 | 64 | 72 |
| **9** | 9 | 18 | 27 | 36 | 45 | 54 | 63 | 72 | 81 |

In [ ]:

**Exercice 12**: Écrire un programme demandant à l'utilisateur de saisir une valeur numérique positive n et affichant toutes les valeurs n, n - 1, . . . , 2, 1, 0.

In [ ]:

**Exercice 13**: Écrire un algorithme demandant la saisie d'un nombre n et calculant $n^n$ .
Par exemple, si l'utilisateur saisit 3, l'algorithme lui affiche
- a) $3^3 = 27$. - b) $3^3 = 3 \times 3 \times 3 = 27$. (plus difficile)

In [ ]:

**Exercice 14**: Écrivez un algorithme saisissant un nombre $n$ et calculant la somme suivante :

$$S = 1 + 2 + 3 + ... + n$$

In [ ]:

**Exercice 15**: Écrivez un algorithme saisissant un nombre n et calculant la somme suivante :

$$S = \frac{1}{1} - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \cdots + \frac{(-1)^{n+1}}{n}$$

In [ ]:

**Exercice 16**: Soit $(u_n)$ la suite définie par $u_n = 0.8 \times n + 1$
Afficher les 100 premiers termes termes de la suites $(u_n)$

In [ ]:

**Exercice 17**: Soit $(u_n)$ la suite définie par $\begin{cases} u_{n+1} = 0.8 \times u_n + 1 \\ u_0 = 2 \end{cases}$
Afficher les 100 premiers termes termes de la suites $(u_n)$

In [ ]:

**Exercice 18**: Soit $(u_n)$ la suite définie par $\begin{cases} u_{n+1} = 2 \times u_n \\ u_0 = 0.1 \end{cases}$
Déterminer $n$ tel que $u_n \geq 384\,400\,000\,000$ (distance terre lune en mm)

In [ ]:

**Exercice 19**: On veut empiler des boîtes cubiques comme indiqué ci-dessous.
On dispose de 500 boites, combien de rangés pourra t on faire?
Écrire un programme pour répondre à cette question.

In [ ]:

# 8 Pour aller plus loin

## 8.1 Les fonctions

Les fonctions se définissent de la façon suivante

```
In [ ]: def nomfonction(variable):
            .......
            return valeurs
```

On peut retourner plusieurs valeurs avec une liste ou un tuple.
exemple:

```
In [ ]: def f(n):
            return n**2

        # pour l'appliquer
        for i in range(11):
            print(f(i))
        # ou avec une expression liste
        [f(i) for i in range(11)]
```

## 8.2 Les fonctions récursives

Que fait le programme suivants?:

```
In [ ]: def u(n):
            if n==0:
                return 1
            elif n==1:
```

```
            return 1
        else: return u(n-1)+u(n-2)

    u(5)
```

Chapitre suivant: Le slicing et les structures de liste

In [ ]: