INFO2

Magistère de Mathématiques de Rennes
2ème année

école normale supérieure rennes

UNIVERSITÉ DE RENNES 1

Organisation

- Enseignants
  - Chargé de cours : David Pichardie
  - Chargé de TP : Yon Fernandez De Retana
- 6 CM / 6 TP, le mardi 8h Ker Lann, en salle de TP
- 2 notes
  - 1 TP noté à rendre 14 jours après la séance de TP3 (TP2+TP3)
  - 1 épreuve finale en salle machine

Objectifs

- Acquérir/renforcer des compétences en programmation

- Acquérir des notions élémentaires de programmation orienté objet

- Comprendre les différences et les points communs entre les langages de programmation : *les concepts de programmation*

Langage support

- Langage de programmation multi-paradigme (objet, fonctionnel, concurrent)

- Points forts

  - une syntaxe épurée

  - des librairies de calculs scientifiques puissantes (numpy, sage)

  - très populaire : vos problèmes de syntaxe trouveront toujours une réponse sur le web…

Factoriel

C

```c
#include<stdio.h>
#include<stdlib.h>

long fact(int n)
{
    if(n == 0)
        return 1;
    else
        return n * fact(n-1);
}

int main(int argc, char *argv[])
{
  int n = atoi(argv[1]);
  for (int i=0; i < n; i++) {
    printf("(%d, %ld)\n", i, fact(i));
  }
  return 0;
}
```

Factoriel

OCaml

```
let rec fact n =
  if n = 0 then 1
  else n * fact (n-1)

let _ =
  let n = int_of_string Sys.argv.(1) in
  for i=0 to n-1 do
    Printf.printf "(%d, %d)\n" i (fact i)
  done
```

Factoriel

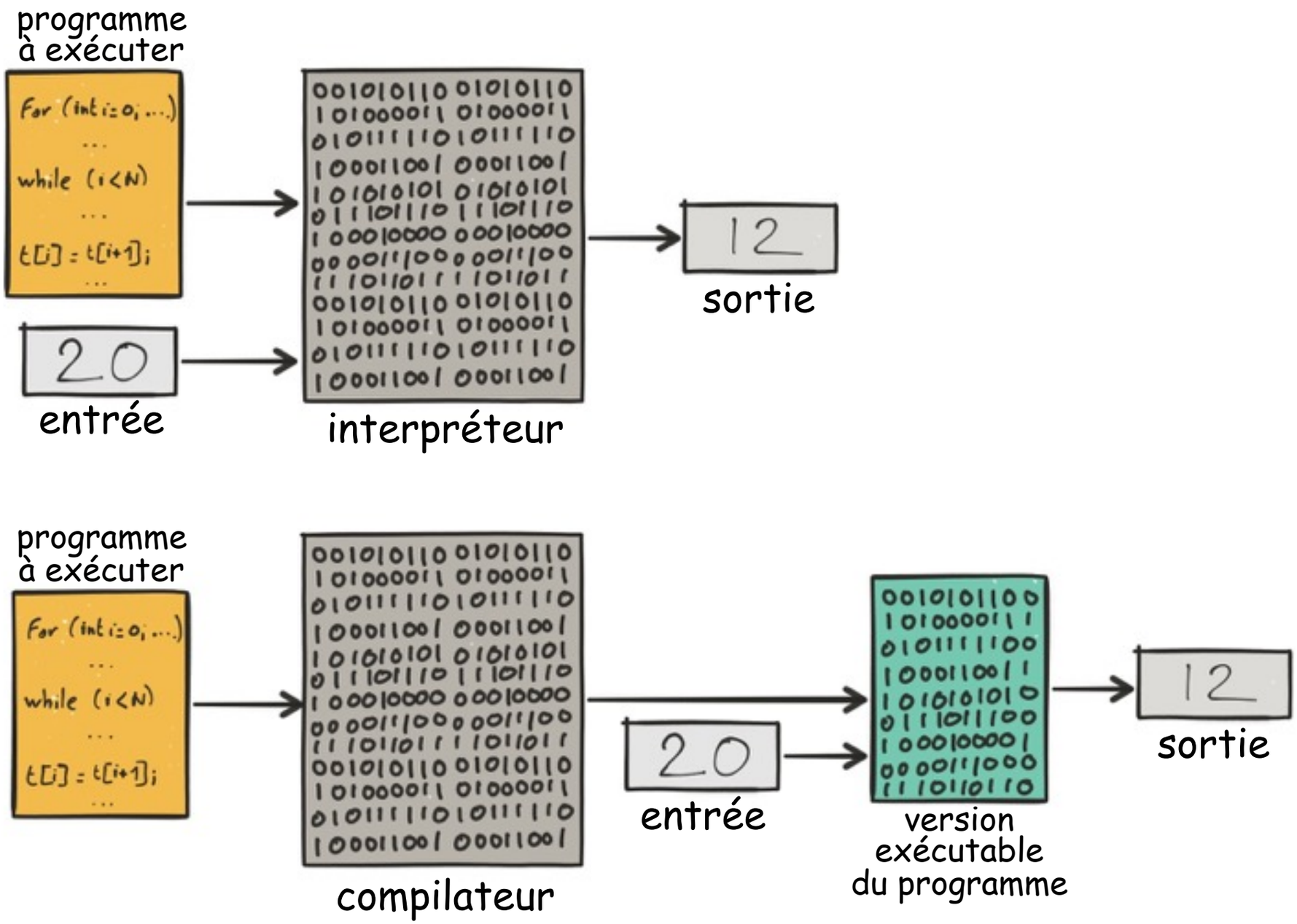
Python

```python
import sys

def fact(n):
    if n == 0:
        return 1
    else:
        return n * fact(n-1)

n = int(sys.argv[1])
for i in range(n):
    print (i,fact(i))
```

Modes d'exécution

- Aucun des trois programmes précédents n'est écrit dans un format directement exécutable par un ordinateur
- Solution 1 : un *interpréteur* exécute le programme
- Solution 2 : un *compilateur* transforme le programme en un programme exécutable

Interpréteur/compilateur



Interpréteur/compilateur

Exemples

Interpréteur

```
$ python fact.py 5
(0, 1)
(1, 1)
(2, 2)
(3, 6)
(4, 24)
```

```
$ ocaml fact.ml 5
(0, 1)
(1, 1)
(2, 2)
(3, 6)
(4, 24)
```

Compilateur

```
$ gcc -o fact fact.c
$ ./fact 5
(0, 1)
(1, 1)
(2, 2)
(3, 6)
(4, 24)
```

```
$ ocamlc -o fact fact.ml
$ ./fact 5
(0, 1)
(1, 1)
(2, 2)
(3, 6)
(4, 24)
```

Python C OCaml

Modes d’exécution comparaison

- Le mode compilé augmente généralement l’efficacité
- Le mode interprété améliore la portabilité
- Un mode hybride : la compilation à la volée

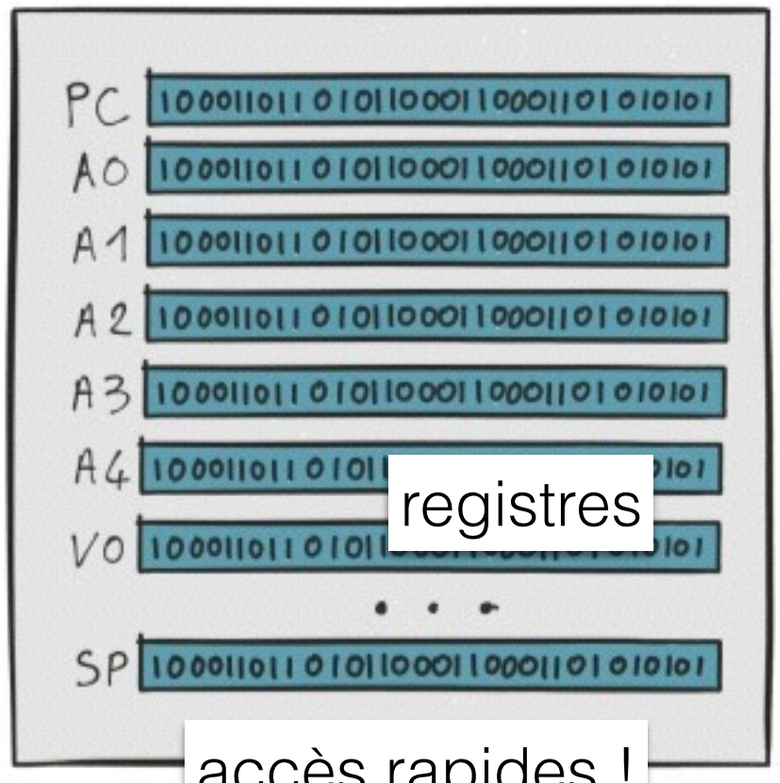
Programme machine ?

Vocabulaire

- Le bit : 0 ou 1
- L’octet : 8 bits
- Le mot : 32 ou 64 bits (selon l’architecture)

Une machine simplifiée

CPU



accès rapides !

+

Mémoire

tableau d'octets,
indice par des mots

code
+
données

accès lents !

Exécution d'un programme machine

- Le registre PC contient l'adresse de la prochaine instruction à exécuter

- Les 4 (ou 8) octets à cette adresse sont lues et décodées comme une instruction

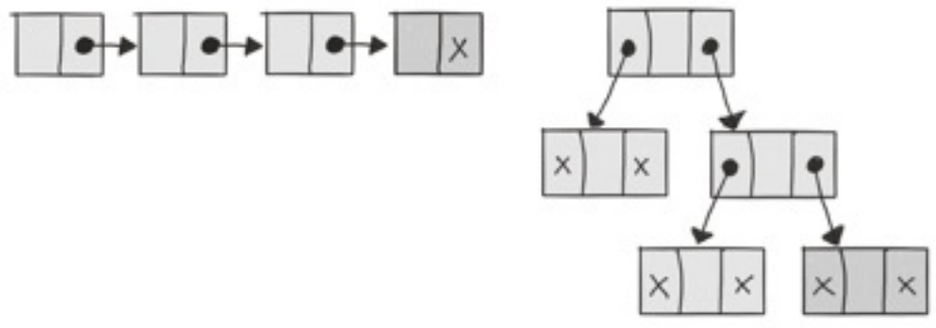
- L'instruction est exécutée (ce qui peut modifier l'état courant des registres et de la mémoire)

- Le registre PC est mis à jour et on recommence…

Les données

- Elles peuvent représenter des nombres dans des formats divers…
  - entiers de capacités limités,
  - un sous-ensemble des nombres réels : les *flottants*
- Mais aussi des données structurées
  - tableaux

  - structures chaînées



Représentations des nombres entiers

```
$ ./fact 30
(0, 1)
(1, 1)
(2, 2)
(3, 6)
(4, 24)
(5, 120)
(6, 720)
(7, 5040)
(8, 40320)
(9, 362880)
(10, 3628800)
(11, 39916800)
(12, 479001600)
(13, 6227020800)
(14, 87178291200)
(15, 1307674368000)
(16, 20922789888000)
(17, 355687428096000)
(18, 6402373705728000)
(19, 121645100408832000)
(20, 2432902008176640000)
(21, -4249290049419214848)
(22, -1250660718674968576)
(23, 8128291617894825984)
(24, -7835185981329244160)
(25, 7034535277573963776)
(26, -1569523520172457984)
(27, -5483646897237262336)
(28, -5968160532966932480)
(29, -7055958792655077376)
```

```
$ ocaml fact.ml 30
(0, 1)
(1, 1)
(2, 2)
(3, 6)
(4, 24)
(5, 120)
(6, 720)
(7, 5040)
(8, 40320)
(9, 362880)
(10, 3628800)
(11, 39916800)
(12, 479001600)
(13, 6227020800)
(14, 87178291200)
(15, 1307674368000)
(16, 20922789888000)
(17, 355687428096000)
(18, 6402373705728000)
(19, 121645100408832000)
(20, 2432902008176640000)
(21, -4249290049419214848)
(22, -1250660718674968576)
(23, -1095080418959949824)
(24, 1388186055525531648)
(25, -2188836759280812032)
(26, -1569523520172457984)
(27, 3739725139617513472)
(28, 3255211503887843328)
(29, 2167413244199698432)
```

```
$ python fact.py 30
(0, 1)
(1, 1)
(2, 2)
(3, 6)
(4, 24)
(5, 120)
(6, 720)
(7, 5040)
(8, 40320)
(9, 362880)
(10, 3628800)
(11, 39916800)
(12, 479001600)
(13, 6227020800)
(14, 87178291200)
(15, 1307674368000)
(16, 20922789888000)
(17, 355687428096000)
(18, 6402373705728000)
(19, 121645100408832000)
(20, 2432902008176640000)
(21, 51090942171709440000L)
(22, 1124000727777607680000L)
(23, 25852016738884976640000L)
(24, 620448401733239439360000L)
(25, 15511210043330985984000000L)
(26, 403291461126605635584000000L)
(27, 10888869450418352160768000000L)
(28, 304888344611713860501504000000L)
(29, 8841761993739701954543616000000L)
```

Représentations des nombres entiers

- Représentation binaire sur *n* bits

$$x = b_{n-1} \cdots b_0$$

- Interprétation non-signée

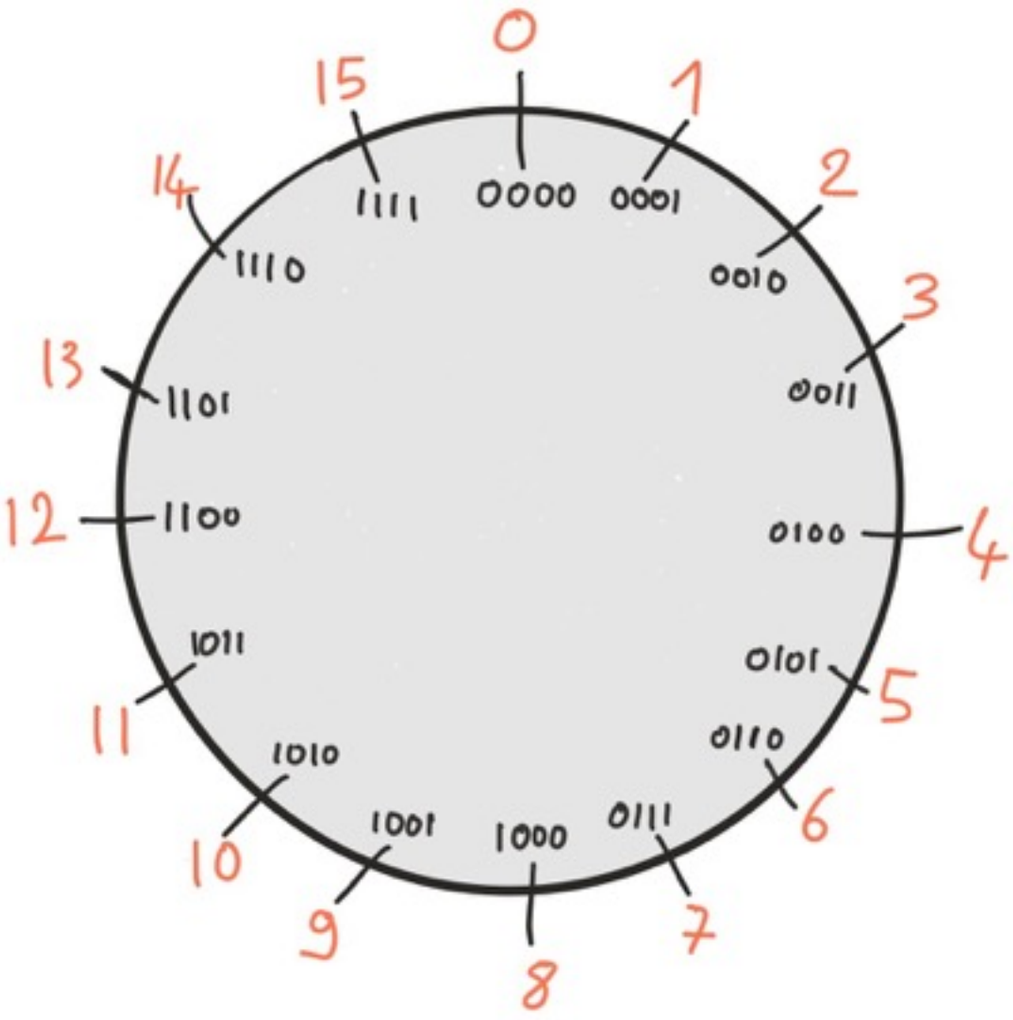
$$U(x) = \sum_{k=0}^{n-1} b_k 2^k$$

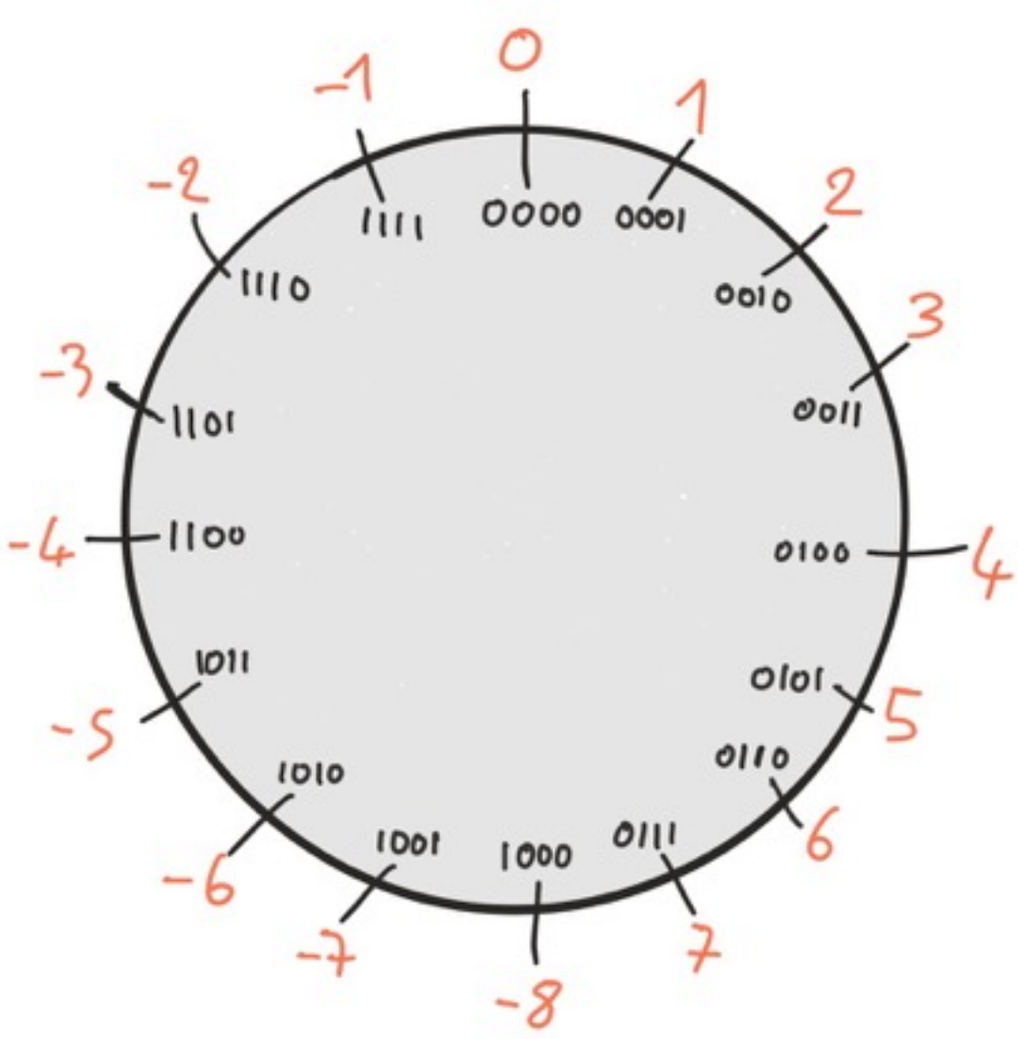
- Interprétation signée

$$S(x) = -b_{n-1} 2^{n-1} + \sum_{k=0}^{n-2} b_k 2^k$$

Représentation des nombres entiers

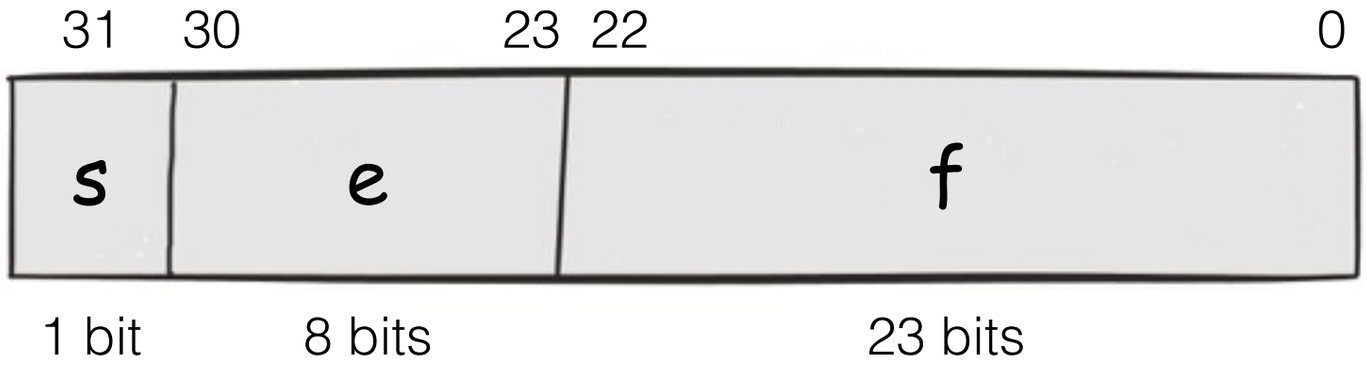
n=4





Les nombres flottants

$x =$



$$F(x) = (-1)^s \times M \times 2^{e-127}$$

$$M = \begin{cases} 1, f_{23} f_{22} \cdots f_0 & e \neq 0, e \neq 255 \\ 0, f_{23} f_{22} \cdots f_0 & e = 0 \end{cases}$$

$$F(x) = -\infty, +\infty \text{ ou Nan si } e = 255$$

Les nombres flottants

vus d'avion

- Quand les calculs réelles ne tombent pas sur un flottant, il faut arrondir vers un flottant proche
  - cela peut occasionner des pertes de précision importantes

Les nombres flottants
exemple de perte de précision

$$r = (x + a) - (x - a) = 2a$$

```
int main () {
    float x, y, z, r;
    x = 1.000000019e+38f;
    y = x + 1.0;
    z = x - 1.0;
    r = y - z;
    printf("r = %f\n", r);
}
```

Affiche 0 !

Les nombres flottants
exemple de perte de précision

$$r = (x + a) - (x - a) = 2a$$

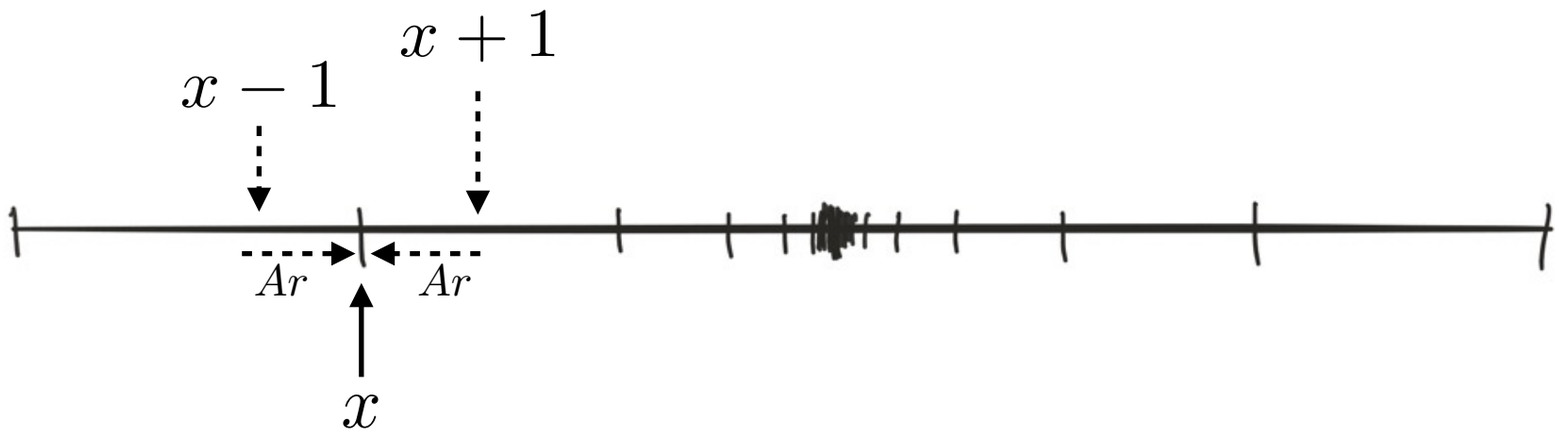
```
int main () {
    float x, y, z, r;
    x = 1.000000019e+38f;
    y = x + 1.0e21f;
    z = x - 1.0e21f;
    r = y - z;
    printf("r = %f\n", r);
}
```

Affiche 0 !

Les nombres flottants
exemple de perte de précision

$$r = (x + a) - (x - a) = 2a$$

```
int main () {
    float x, y, z, r;
    x = 1.000000019e+38f;
    y = x + 1.0e21f;
    z = x - 1.0e21f;
    r = y - z;
    printf("r = %f\n", r);
}
```



$$x +_f 1 = Ar(x + 1) = x$$

$$x -_f 1 = Ar(x - 1) = x$$

Récapitulatif

- Les programmes sont exécutés selon deux modes d'exécution : compilation ou interprétation

- Le langage machine s'appuie sur la notion de mots de bits et suit un algorithme d'exécution extrêmement simple. Le compilateur est en charge de traduire nos programmes vers cette représentation.

- Les données sont représentées par des séquences de bits.

- Les entiers machines ont une capacité limité. Python fait le choix de proposer une précision arbitraire, quitte à devoir occuper plus de mémoire.

- Les nombres réels sont généralement représentés par un sous-ensemble des rationnels : les flottants. Les pertes de précision par rapport aux opérations mathématiques réelles peuvent être très grandes.

Exercice

- Ecrire un programme python **`tobin.py`** qui transforme son entrée entière en une représentation binaire sur la ligne de commande
  exemple : `%python tobin.py 11`
  `1 0 1 1`
- Consignes
  - ne pas utiliser de structure auxiliaire type tableau
  - proposer une version avec récursivité, puis sans
- Remarque : pour afficher un caractère sans revenir à la ligne, utiliser la commande **`print`** ***`chaine`*****`,`**

Solution

version avec récursivité

```
import sys

def tobin(n):
    if n<=1: print(n),
    else:
        tobin(n//2)
        print(n % 2),


tobin(int(sys.argv[1]))
print "" #pour revenir a la ligne
```

Solution

version sans récursivité

```
import sys

def tobin(n):
    # on calcule la plus grande puissance de 2 inferieur ou
    # egale a n pour trouver le bit de poid fort
    v = 1
    while v <= n//2 :
        v *= 2 # equivalent a v = v*2

    # puis on parcours les bits de n en partant du plus fort
    while v > 0:
        if n >= v: # v=2**k et le k[eme] bit de n vaut 1
            print '1',
            n = n-v
        else: # v=2**k et le k[eme] bit de n vaut 0
            print '0',
        v = v//2


tobin(int(sys.argv[1]))
print "" #pour revenir a la ligne
```