

Informatique et Programmation

Cours 1

Jean-Jacques Lévy

`jean-jacques.levy@inria.fr`

`http://jeanjacqueslevy.net/prog-py`

Plan

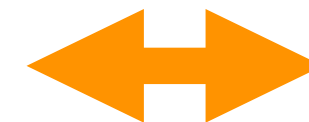
- machines informatiques
- systèmes informatiques
- programmes
- langages de programmation
- la programmation
- un langage pour le cours: Python 3
- premiers programmes

dès maintenant: **télécharger Python 3 en** `http://www.python.org`

Machines informatiques

- les premiers vrais ordinateurs en 1960: IBM 704, 7040, 360/370, ...
- le langage des machines est exprimé en **binaire**: 0 ou 1 [pour des raisons électriques]

```
mac$ od -x a.out | head -4
00000000      0162      0006      8f30      2e89      6000      0000      0060      0000
00000020      0038      0007      010b      020a      4000      0000      2000      0000
00000040      0000      0000      0140      0040      0000      0040      0000      1000
00000060      2000      1000      fffe      f3ff      0000      0000      1013      0000
mac$ od -x a.out | tail -5
01164000      0000      0028      0216      0000      3030      0040      1046      0000
01164020      0000      0023      021c      0000      1580      1000      f0c5      ffff
01164040      0000      0023      0220      0000      12e0      1000      f341      ffff
01164060      0000      0023      0228      0000      12e4      1000      f341      ffff
```



```
mac$ cat t.c
#include <stdio.h>

char s[100] = "voici une chaîne de caractères" ;

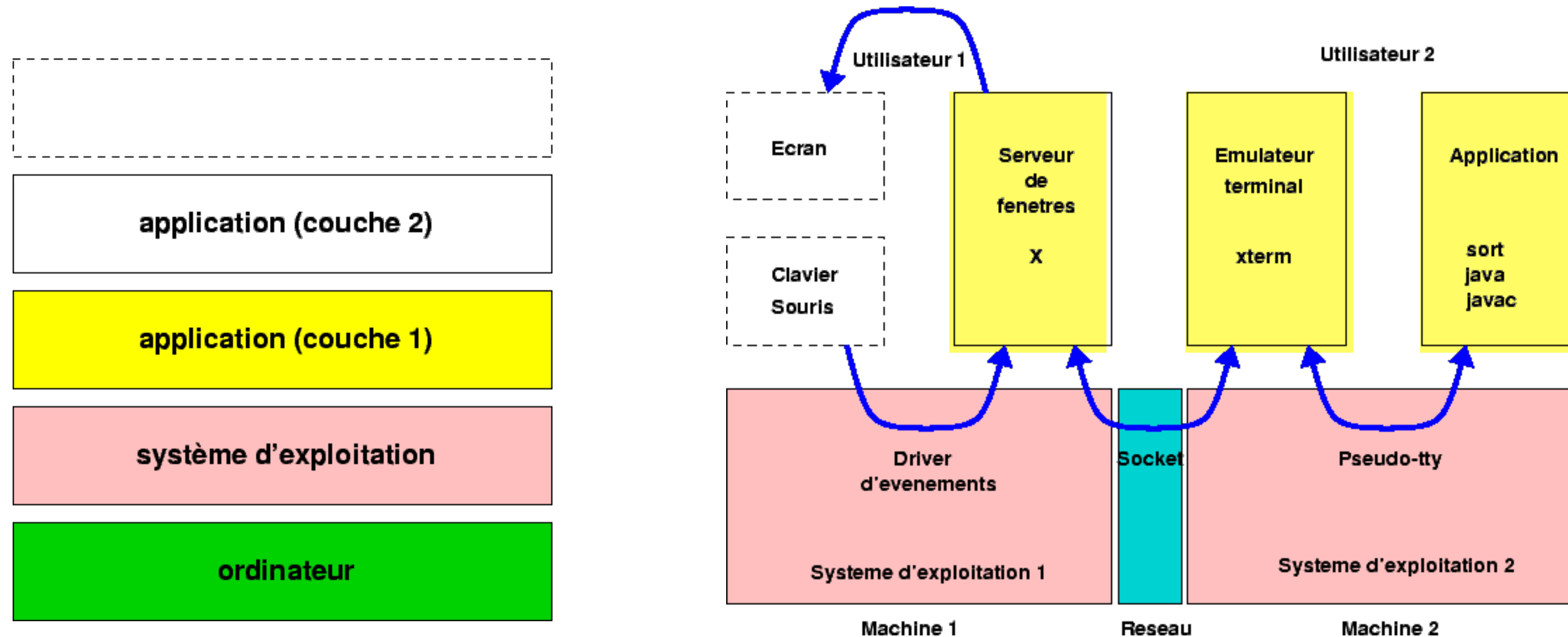
main()
{
    printf ("%s\n", s);
}
```

- ici 40256 octets = 322048 bits
- dans ce binaire, il y a des commandes [opérations à effectuer] et des données
- ce binaire est chargé dans la mémoire de l'ordinateur
- et l'ordinateur exécute les commandes du binaire
- chaque opération prend quelques micro-secondes

*bits = binary digit
octets = 8 bits = bytes*

Systemes informatiques

- Windows, MacOS, Unix, Linux, ios, android, GFS, ...
- faire marcher l'ordinateur avec ses périphériques (écran, disques, clavier, souris, utilisateurs, réseau, ...)
- les OS sont d'une complexité extrême (~100 millions de lignes de code)

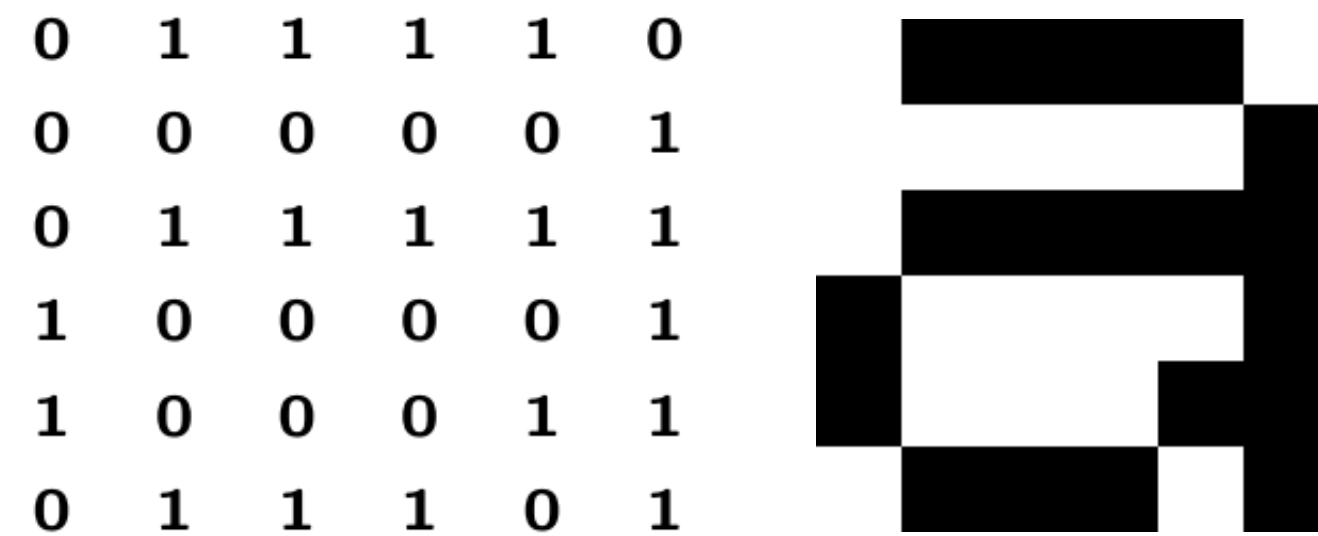


*OS = operating system
= système d'exploitation*

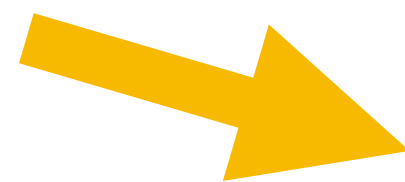
- au 15ème siècle, on construisait des cathédrales
- au 20ème siècle, on a construit des systèmes informatiques

Exemples de problèmes non triviaux

- afficher des lettres sur l'écran
- dessiner rapidement des figures sur l'écran (vecteurs, cercles, ovales, *splines*)
- écrire efficacement sur les disques
- ne pas perdre les messages envoyés ou reçus sur le réseau
- sauvegarder les données (*backups*)



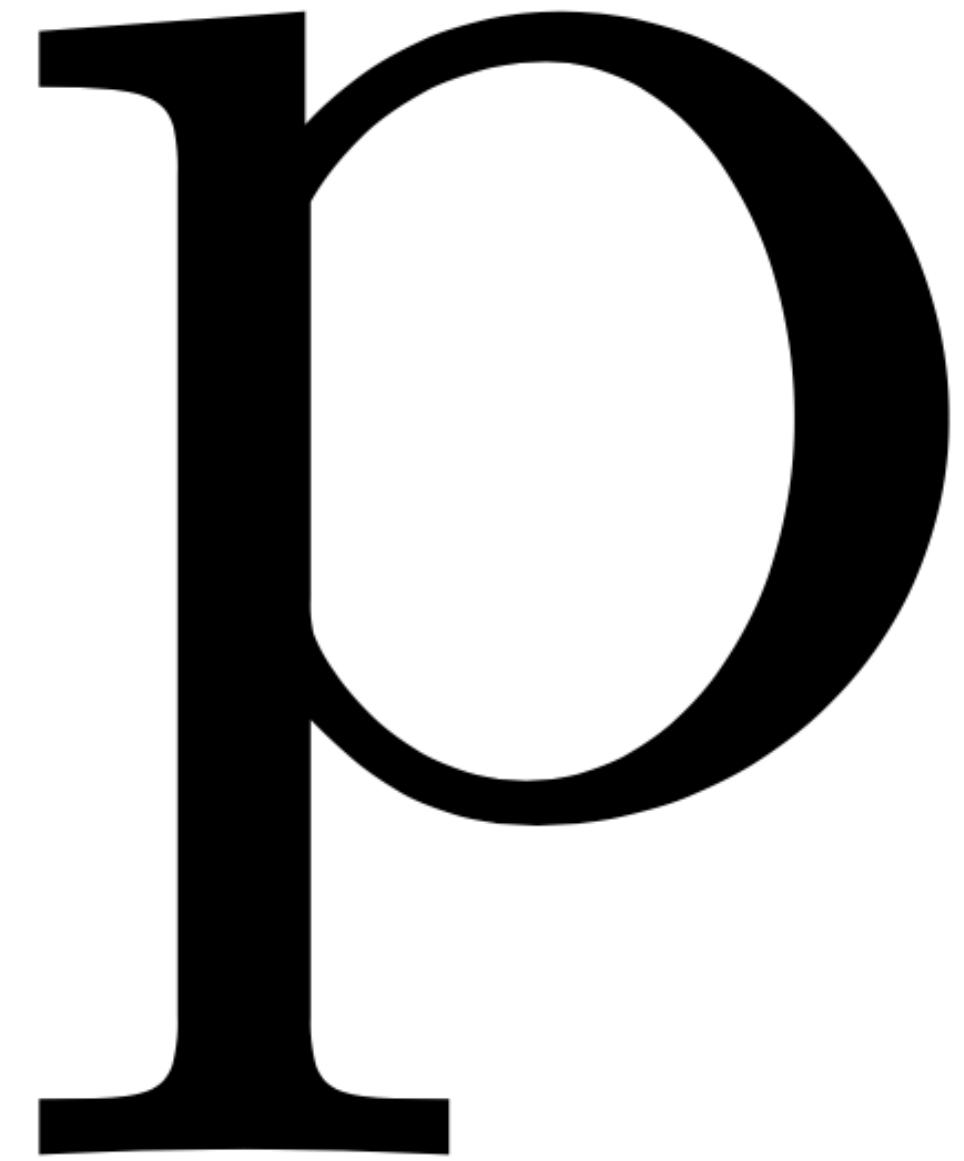
lettre 'a' = tableau de 0 et 1



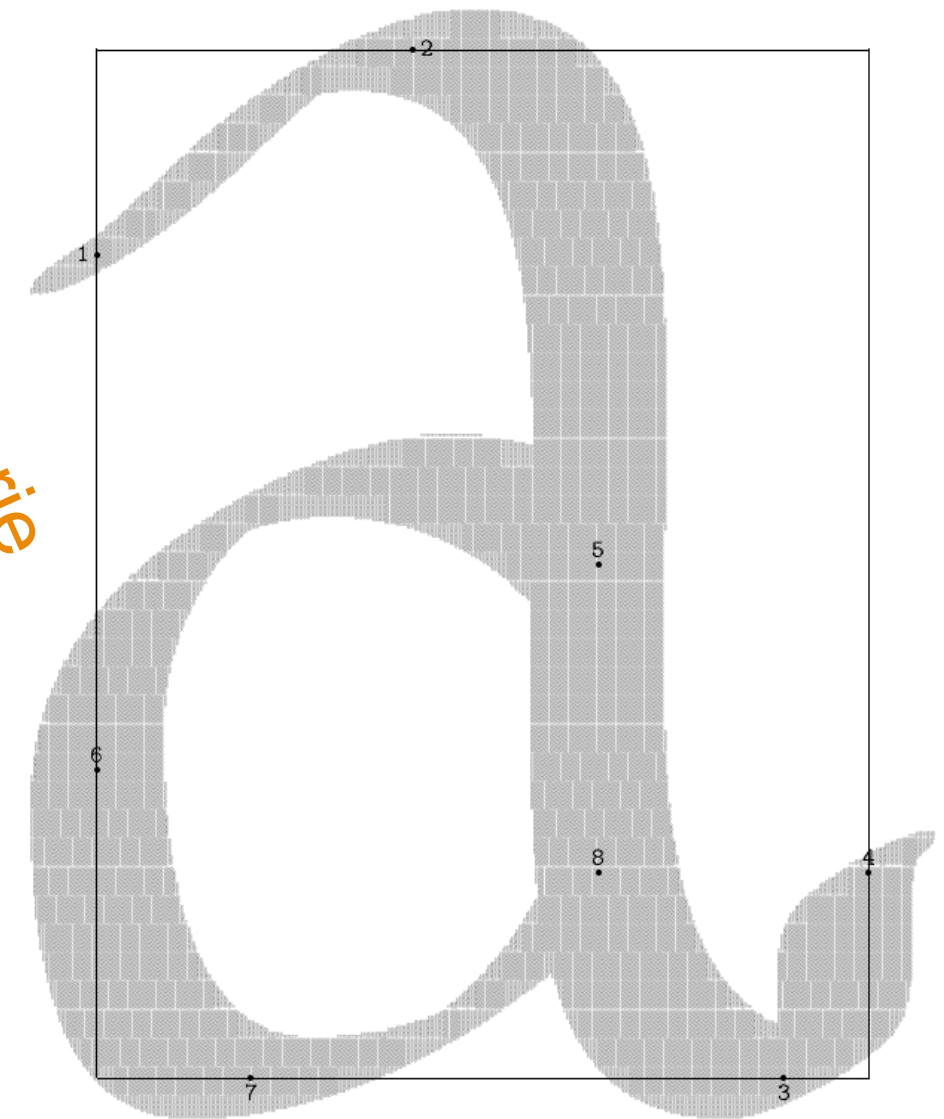
écran *bitmap*

écran = grand tableau de 0 et 1 (taille = *résolution* de l'écran)

metafont = réinvention de l'imprimerie



METAFONT output 2012.05.07:1643 Page 1 Character 97



Programmes

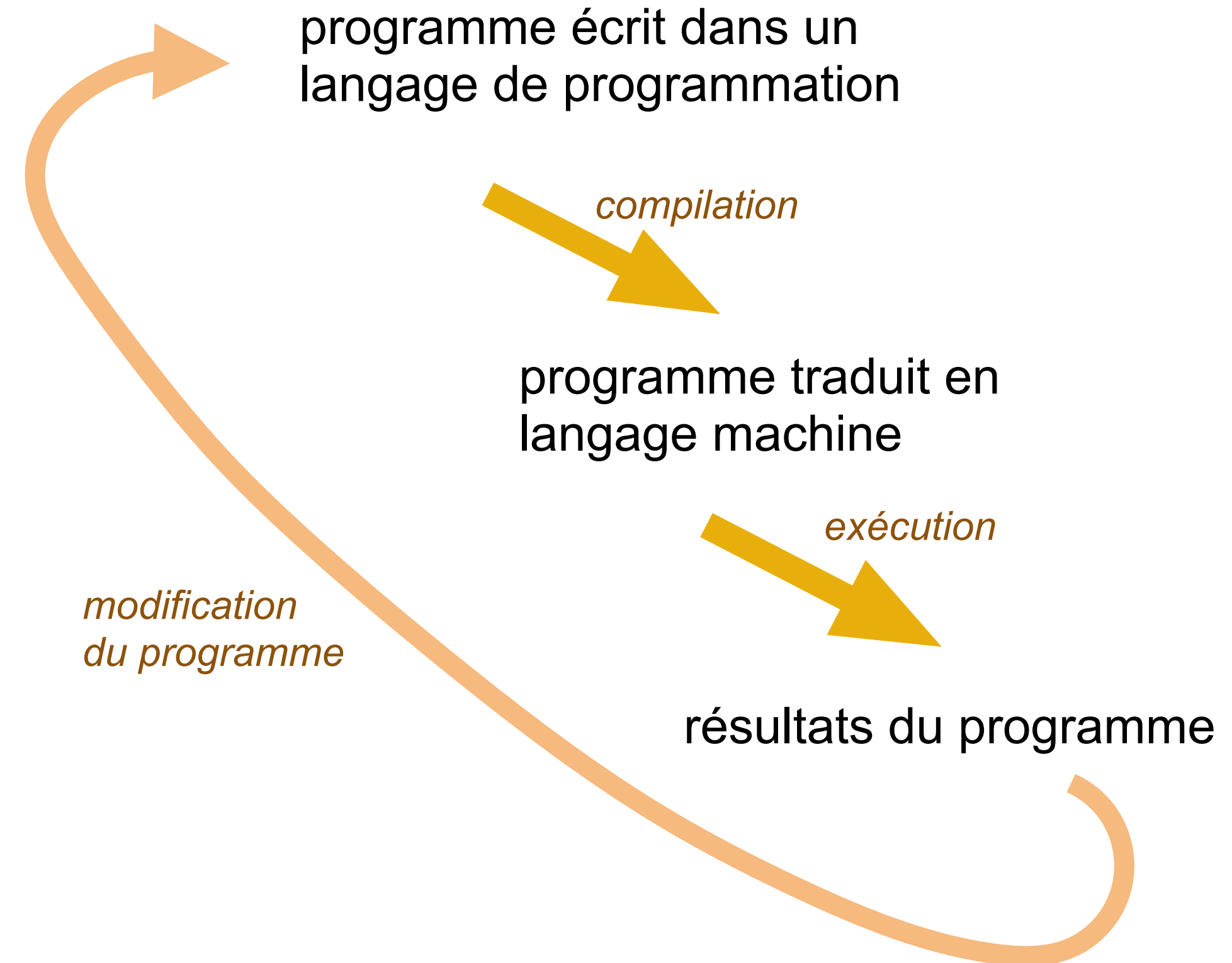
- **résoudre** un problème ou **rechercher** ou **communiquer** ou **calculer**
- séquence d'instructions
- avec des données
- rendre les résultats sur écran ou sur les disques
- l'ordinateur est très **rapide** et ne fatigue jamais
- un programme a souvent **beaucoup de données**
- il peut devenir complexe et il faut **maîtriser** cette complexité
- un programme est très **précis**
- un programme peut contenir des **erreurs**
- on doit le tester pour éviter ces erreurs
- il faut **réfléchir** avant de se lancer dans la programmation

Langages de programmation

- raisonner dans le langage machine (binaire) est une mission impossible
- le langage de la programmation doit être compréhensible
- les langages de programmation sont plus ou moins éloignés du langage machine
- langages de bas-niveau: assembleur, le langage C
- langages de haut niveau: Algol, Pascal, C++, Lisp, Ocaml, Haskell, Java, Python, ..
- il y a 100 langages possibles selon le style de programmation voulue
- c'est aussi une question de goût
- **mais** les bases de la programmation sont **universelles**
- les **algorithmes** et les **structures de données** sont indépendants du langage de programmation
- et la **structure des programmes** est aussi souvent universelle

La programmation

- conception de la structure du programme, des algorithmes et des structures de données
- conception si possible en petits modules
- écriture du programme
- le compiler et l'exécuter
- le tester sur un bon jeu de tests



Un langage pour le cours: Python 3

- Python est un langage **moderne** de haut-niveau
- Python est un langage orienté-objet
- Python a un mode **interactif** commode pour la mise au point des programmes
- Python a un **typage dynamique**
- Python a un grand nombre de librairies car il est populaire chez les mathématiciens
- le cours utilise Python 3
- on télécharge Python 3 en `http://www.python.org`
- un bon tutorial Python se trouve en `http://www.programiz.com/python-programming`

Débuter en Python

- utiliser un système intégré (Visual Studio ou autre)
- ou utiliser une simple fenêtre terminal [le plus rapide pour démarrer]
- avec un éditeur de texte (Emacs, VI, TextEdit, ..)
- sur la fenêtre terminal, on tape:

```
mac$ python
Python 3.7.9 (default, Sep  6 2020, 13:20:25)
[Clang 11.0.3 (clang-1103.0.32.62)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- et on peut fonctionner en mode calculette:

```
>>> 23 + 42
65

>>> 438 * 234
102492
>>> (438 * 234) + 35
102527
>>> ((438 * 234) + 35 / 3)
102503.66666666667
```

Débuter en Python

- mode calculette

```
>>> x = 45
>>> 3 * x + 2
137
```

- avec des chaînes de caractères

```
>>> print ("Bonjour les amis !")
Bonjour les amis !
>>> y = "Bonjour les amis"
>>> y + " et la famille"
'Bonjour les amis et la famille'
>>> z = " !"
>>> y + " et la famille" + z
'Bonjour les amis et la famille !'
```

```
>>> s = "Bonjour les amis !"
>>> s[0]
'B'
>>> s[1]
'o'
>>> s[2]
'n'
>>> s[9]
'e'
>>> s[-1]
'!'
>>> s[-2]
','
>>> s[-3]
's'
```


Débuter en Python

- définir des fonctions

```
>>> def double (x) :  
...     return (2 * x)  
...  
>>> double(3)  
6
```

```
>>> def concatene (s1, s2) :  
...     return (s1 + " " + s2)  
...  
>>> concatene ("Hello", "World !")  
'Hello World !'
```

Premiers programmes

Le calcul de la date de Pâques

Premier dimanche après la première lune
qui suit l'équinoxe de printemps.

Soit Y l'année, dont on cherche la date de Pâques.

1. **Golden number** $G = (Y \bmod 19) + 1$
2. **Century** $C = \lfloor Y/100 \rfloor + 1$
3. **Corrections** $X = \lfloor 3C/4 \rfloor - 12, Z = \lfloor (8C + 5)/25 \rfloor - 5$
4. **Find Sunday** $D = \lfloor 5Y/4 \rfloor - X - 10$
5. **Epact** $E = (11G + 20 + Z - X) \bmod 30$.
Si $E=25$ et $G>11$, ou si $E=24$, alors $E \leftarrow E+1$
6. **Find full moon** $N = 44 - E$. Si $N < 21$, alors $N \leftarrow N + 30$
7. **Advance to Sunday** $N \leftarrow N + 7 - ((D + N) \bmod 7)$
8. **Getmonth** Si $N > 31$, la date est le $(N - 31)$ AVRIL
Sinon, la date est le N MARS.

```
>>> def paques (y) :  
...     g = (y % 19) + 1  
...     c = y // 100 + 1  
...     x = 3 * c // 4 - 12  
...     z = (8 * c + 5) // 25 - 5  
...     d = 5 * y // 4 - x - 10  
...     e = (11 * g + 20 + z - x) % 30  
...     if e == 25 and g > 11 or e == 24 :  
...         e = e + 1  
...     n = 44 - e  
...     if n < 21 :  
...         n = n + 30  
...     j = n + 7 - ((d + n) % 7)  
...     if j > 31 :  
...         print (str(j - 31) + " avril")  
...     else :  
...         print (str(j) + " mars")  
...  
>>>  
>>>  
>>> paques (2021)  
4 avril  
>>> paques (2020)  
12 avril  
>>> paques (2024)  
31 mars
```

http://fr.wikipedia.org/wiki/Calcul_de_la_date_de_Pâques

Premiers programmes

- impression

```
>>> def concat_print0 (s1, s2):  
...     print (s1 + " " + s2)  
...  
>>> concat_print0 ("Hello", "World !")  
Hello World !
```

```
>>> def concat_print1 (s, n, x):  
...     print ("%s vaut %d ou %.2f" %(s, n, x))  
...  
>>> concat_print1 ("Le resultat", 32, 28.5)  
Le resultat vaut 32 ou 28.50
```

```
>>> def concat_print2 (s, n, x):  
...     print ("{} vaut {} ou {}".format (s, n, x))  
...  
>>> concat_print2 ("Le resultat", 32, 28.5)  
Le resultat vaut 32 ou 28.5
```


Premiers programmes

Le calcul de la date de Pâques

Premier dimanche après la première lune
qui suit l'équinoxe de printemps.

Soit Y l'année, dont on cherche la date de Pâques.

1. **Golden number** $G = (Y \bmod 19) + 1$
2. **Century** $C = \lfloor Y/100 \rfloor + 1$
3. **Corrections** $X = \lfloor 3C/4 \rfloor - 12, Z = \lfloor (8C + 5)/25 \rfloor - 5$
4. **Find Sunday** $D = \lfloor 5Y/4 \rfloor - X - 10$
5. **Epact** $E = (11G + 20 + Z - X) \bmod 30$.
Si $E=25$ et $G>11$, ou si $E=24$, alors $E \leftarrow E+1$
6. **Find full moon** $N = 44 - E$. Si $N < 21$, alors $N \leftarrow N + 30$
7. **Advance to Sunday** $N \leftarrow N + 7 - ((D + N) \bmod 7)$
8. **Getmonth** Si $N > 31$, la date est le $(N - 31)$ AVRIL
Sinon, la date est le N MARS.

```
>>> def Paques (y) :  
...     g = (y % 19) + 1  
...     c = y // 100 + 1  
...     x = 3 * c // 4 - 12  
...     z = (8 * c + 5) // 25 - 5  
...     d = 5 * y // 4 - x - 10  
...     e = (11 * g + 20 + z - x) % 30  
...     if e == 25 and g > 11 or e == 24 :  
...         e = e + 1  
...     n = 44 - e  
...     if n < 21 :  
...         n = n + 30  
...     j = n + 7 - ((d + n) % 7)  
...     if j > 31 :  
...         print ("%d avril %d" %(j - 31, y))  
...     else :  
...         print ("%d mars %d" %(j, y))  
...  
  
>>> Paques (2020)  
12 avril 2020  
>>> Paques (2021)  
4 avril 2021  
>>> Paques (2024)  
31 mars 2024
```

http://fr.wikipedia.org/wiki/Calcul_de_la_date_de_Pâques

Premiers programmes

- constantes littérales: nombres (entiers `int`, flottants `float`, complexes `complex`)

```
>>> a = 5
>>> print(a, "is of type", type(a))
5 is of type <class 'int'>
```

```
>>> a = 2.0
>>> print(a, "is of type", type(a))
2.0 is of type <class 'float'>
```

```
>>> a = 1+2j
>>> print(a, "is complex number?", isinstance(1+2j,complex))
(1+2j) is complex number? True
```

- constantes littérales: chaînes de caractères `string`

```
>>> s = "This is a string"
>>> print(s)
This is a string
```

```
>>> s = '''A multiline
... string'''
>>> print(s)
A multiline
string
```

- constantes littérales: booléens `bool`

```
>>> True
True
>>> type (True)
<class 'bool'>
>>> type (False)
<class 'bool'>
```

Premiers programmes

- quelques programmes sur des données scalaires

```
>>> import math
>>>
>>> def surface (r) :
...     print ("%2f" %(4 * math.pi * r ** 2))
...
>>> surface (3)
113.10
>>> surface (1.3)
21.24
```

- calculer la surface d'un carré ou triangle
- calculer le volume d'un cube ou sphère

Premiers programmes

- opérations de comparaison

```
>>> 3 == 4
False
>>> 3 != 4
True
>>> 3 < 4
True
>>> 3 <= 4
True
>>> True and False
False
>>> True or False
True
```

- expressions booléennes composites

```
>>> a = 4
>>> b = 5
>>> a > 8 and b < 16
False
>>> not a > 8
True
```

- itération while

```
>>> i = 0
>>> while i < 10 :
...     i = i + 1
...     print (i)
...
1
2
3
4
5
6
7
8
9
10
```


Premiers programmes

- calcul du PGCD (algorithme d'Euclide) de deux nombres entiers m et n
- écrire la date de Pâques de 2021 à 2050
- trouver la première année où le calcul de la date de Pâques est incorrect
- calculer la suite de Syracuse. Vérifier son résultat jusqu'à 1000

$$u_{n+1} = \begin{cases} 3u_n + 1 & \text{si } u_n \text{ impair} \\ u_n/2 & \text{si } u_n \text{ pair} \end{cases}$$

Prochain cours

- structures de données composites
- classes et objets