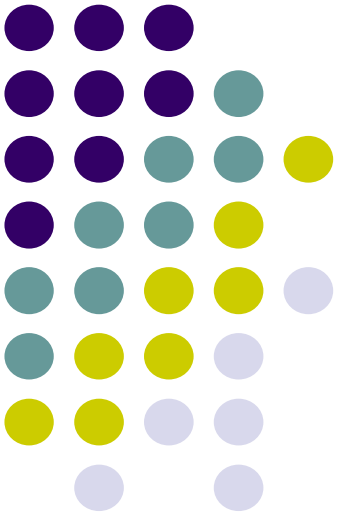
MI-GLSD-M1 -UEM213 : Paradigmes de langages de Programmation

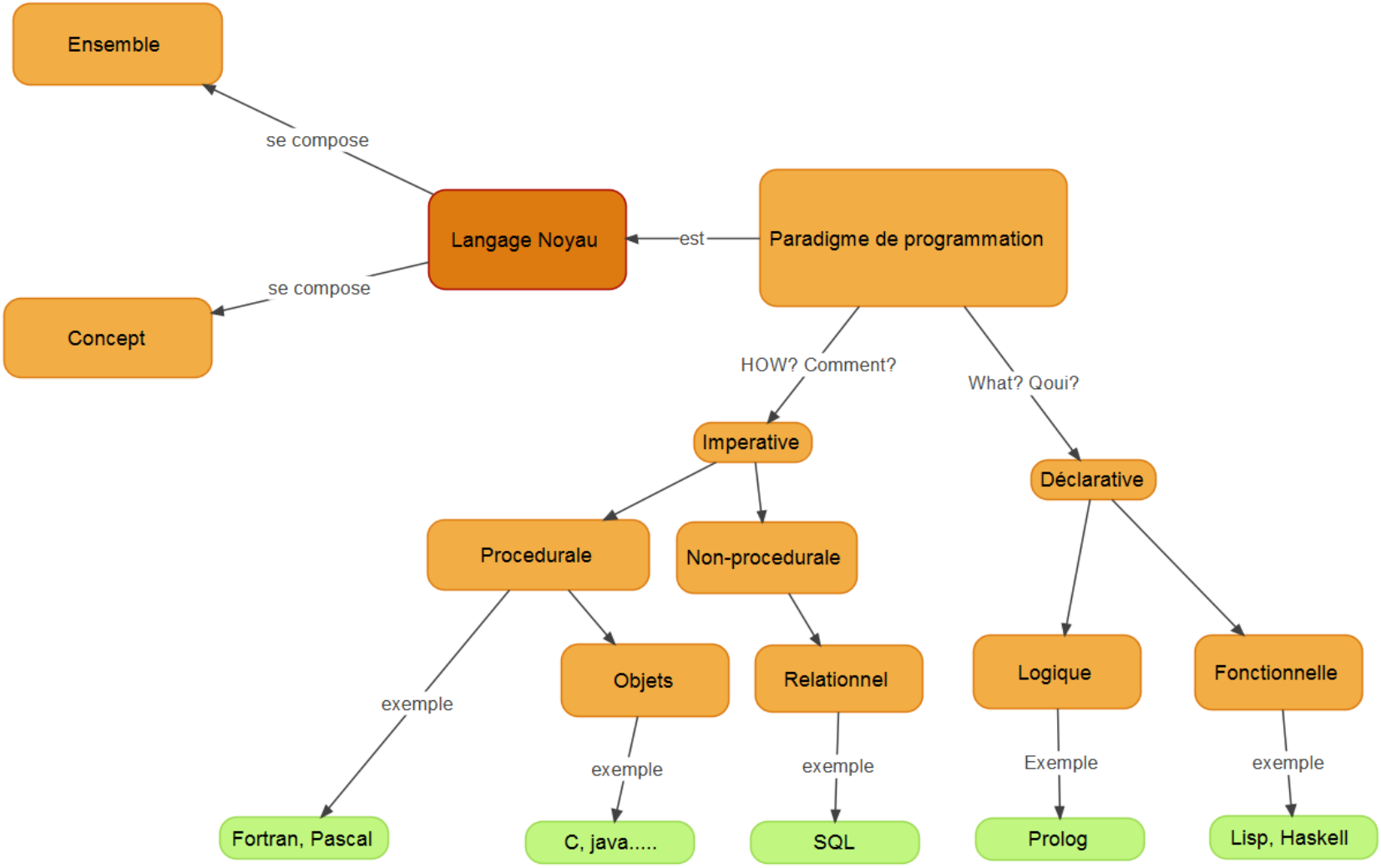
Chapitre II: Paradigme impératif

A. HARICHE
Université de Djilali Bounaama, Khemis Meliana (UDBKM)
Faculté des sciences et de la technologie
Département de mathématiques et d'informatique

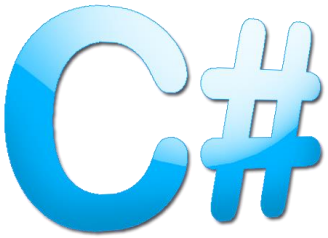
a.hariche@univ-dbkm.dz

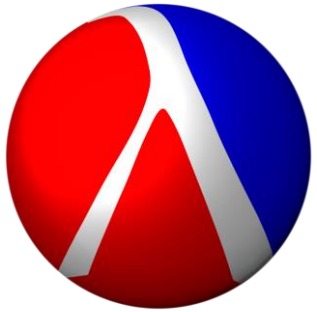


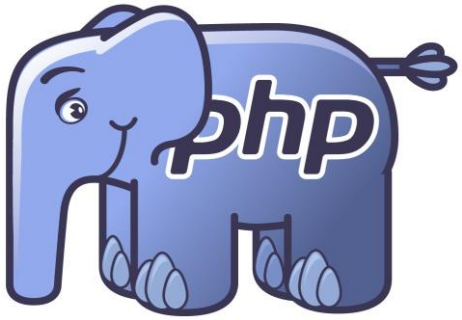
Paradigmes des langages de programmation

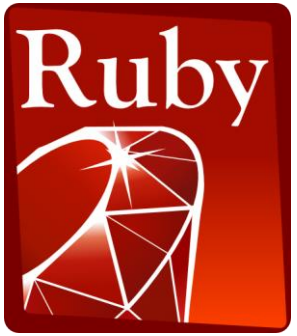


Des centaines de langages de programmation sont utilisés...



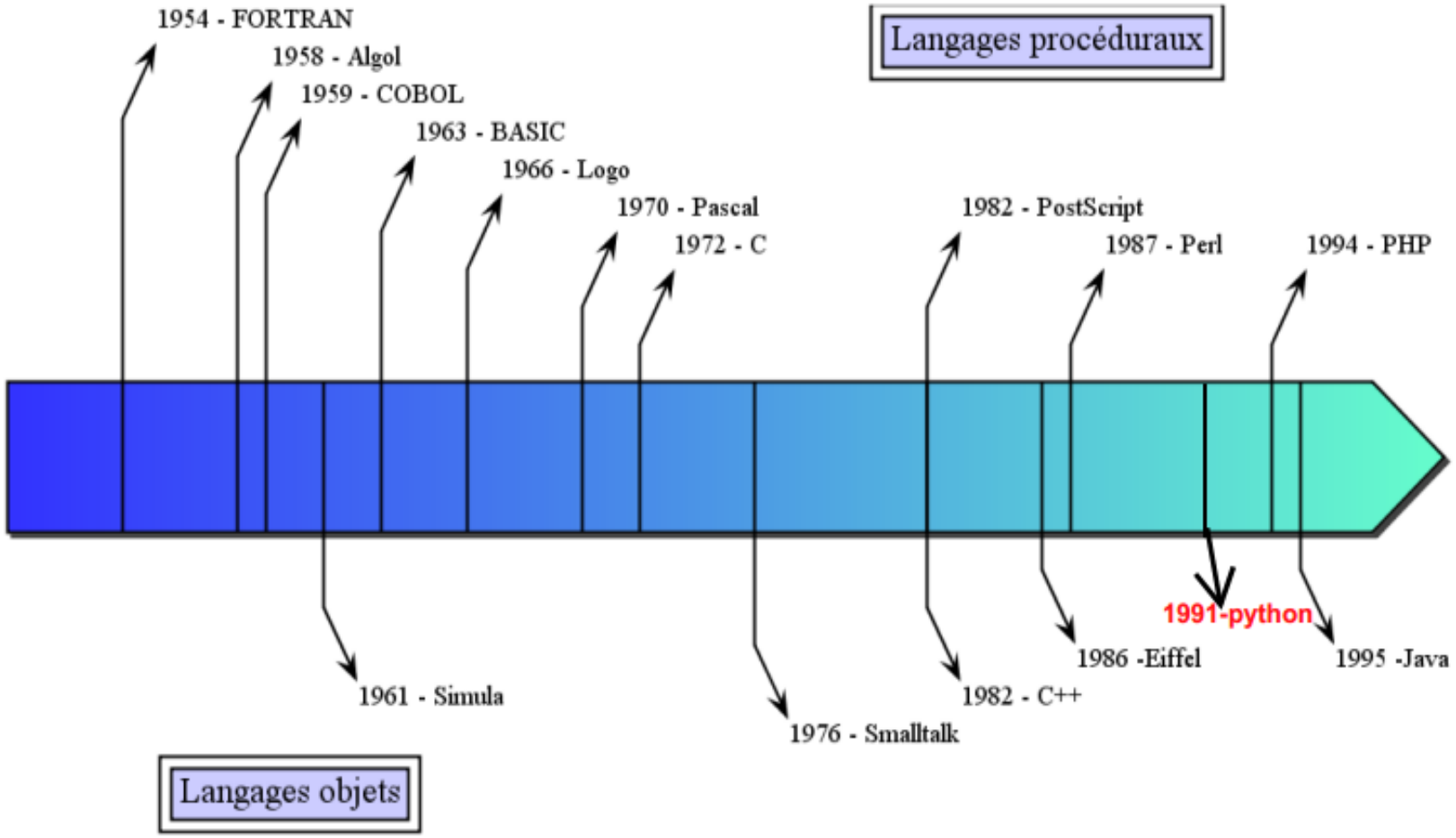




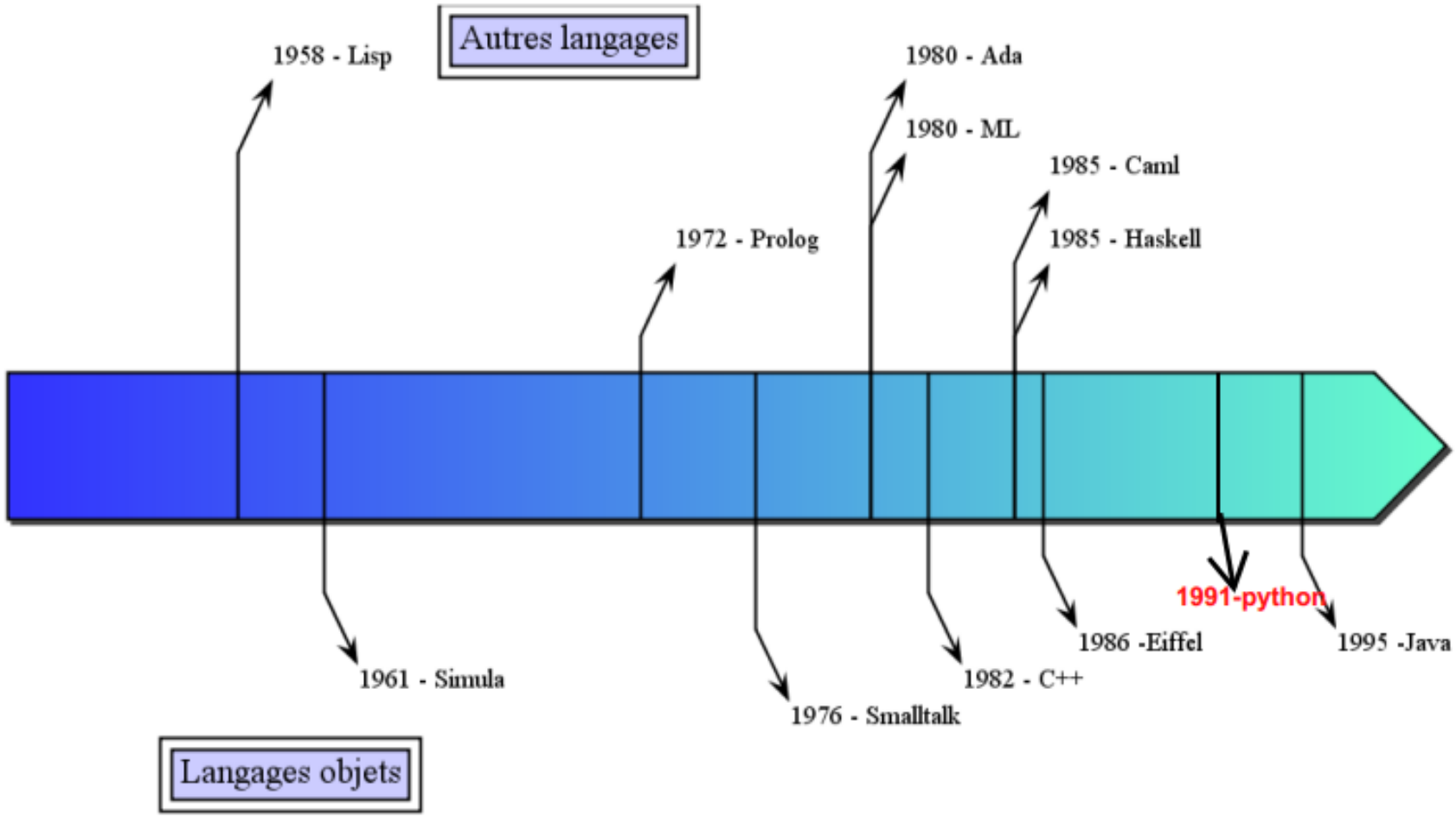


A.HARICHE a.hariche@univ-dbkm.dz

Langages de programmation Historique



Langages de programmation Historique



Un paradigme très reconnu ….

Avantages	Inconvénients
Bien lisible	Le code devient très rapidement volumineux et perd en clarté
Comparativement facile à apprendre	Risque d'erreurs supérieur lors de retouches
Modèle de pensée aisément compréhensible pour les débutants (étapes de résolution)	Les opérations de maintenance bloquent le développement d'applications, car la programmation est étroitement liée au système
Permet de tenir compte des caractéristiques de cas d'application spéciaux	Optimisations et extensions plus difficiles

Peu d'histoire

Church (1930) et Turing (1936)

- fonctions définissables en λ-calcul (programmation fonctionnelle)
  = fonctions récursives (programmation fonctionnelle)
  = machines de Turing (programmation impérative)
- idée de calcul effectif, formalisation de la calculabilité
- machine de Turing = modèle "réaliste" d'ordinateur
- pas encore d'ordinateur construit !
- existence de fonctions non calculables

A.HARICHE a.hariche@univ-dbkm.dz

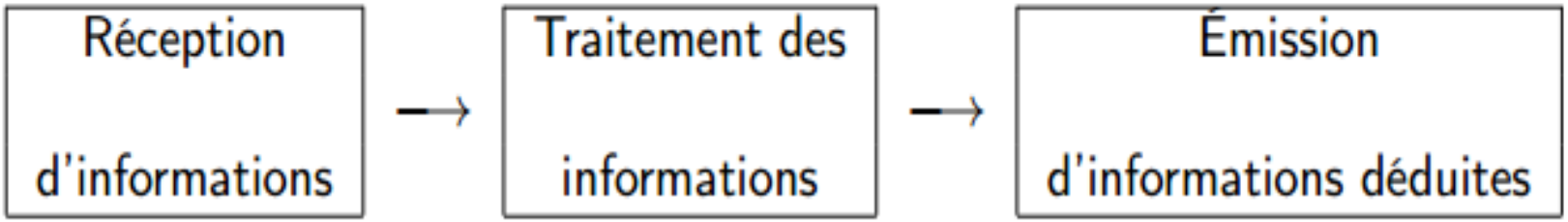
Rappel

Notions supposées connues :

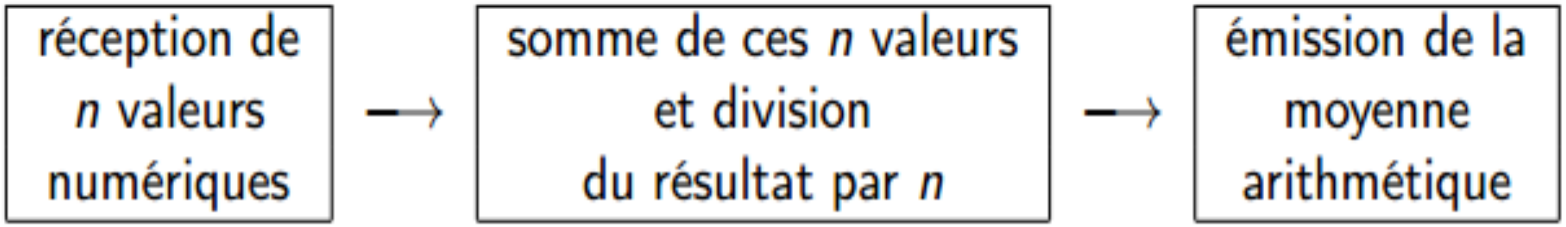
- Notion de programme
- Notion de variable, de type, déclaration
- Instruction conditionnelle
  - `if (...) {...} else {...}`
- Instruction itérative :
  - `while (...) {...}`
  - `do {...} while (...);`
  - `for (...; ...; ...) {...}`
- Notion de fonction, appel de fonction.

Traitement de l'information

Le schéma global d'une application informatique est toujours le même :



Exemple :



Un programme

Tout traitement demandé à la machine, par l'utilisateur, est effectué par l'exécution séquencée d'opérations appelées **instructions**. Une suite d'instructions est appelée un **programme**.

Retenir

*Un programme est une **suite d'instructions** permettant à une système informatique d'exécuter une tâche donnée*

écrit dans un langage de programmation compréhensible (directement ou indirectement) par un ordinateur.

Un programme impératif

Définition (Rappel)

*un **programme impératif** est une séquence d'**instructions** qui spécifie étape par étape les opérations à effectuer pour obtenir à partir des entrées un résultat (la sortie).*

A.HARICHE a.hariche@univ-dbkm.dz

Les fondations

- Les langages impératifs comportent tous ces instructions de bases :
- **Assignation** (ou affectation) : permet de stocker en mémoire (dans une variable) le résultat d'une opération.
- **Condition** : permet d'exécuter un bloc d'instructions si une condition prédéterminée est réalisée.
- **Boucle** : permet de répéter un bloc d'instructions un nombre prédéfini de fois ou jusqu'à ce qu'une condition soit remplie.
- **Branchement** : permet à la séquence d'exécution d'être transférée ailleurs dans le programme (goto).
- **Séquence d'instructions** : désigne le fait d'exécuter, en séquence, plusieurs des instructions ci-dessus.

A.HARICHE a.hariche@univ-dbkm.dz

Un programme impératif Exemple

- On modifie des *variables* par des *instructions*
- Les variables sont des espaces mémoire
- Les instructions sont organisées séquentiellement
- La machine se programme par *effets de bords* (= modifications de l'état général de la machine (mémoire, vidéo, etc.))

Exemple

```
unsigned long int fact(unsigned int n) {
  unsigned long int r=1UL;
  for (; n>0u; --n)
    r *= n;
  return r;
}
```


A.HARICHE a.hariche@univ-dbkm.dz

Un programme impératif Exemple

- On modifie des *variables* par des *instructions*
- Les variables sont des espaces mémoire
- Les instructions sont organisées séquentiellement
- La machine se programme par *effets de bords* (= modifications de l'état général de la machine (mémoire, vidéo, etc.))

Un espace mémoire

Exemple

```
unsigned long int fact(unsigned int n) {
  unsigned long int r=1UL;
  for (; n>0u; --n)
    r *= n;
  return r;
}
```


A.HARICHE a.hariche@univ-dbkm.dz

Un programme impératif Exemple

- On modifie des *variables* par des *instructions*
- Les variables sont des espaces mémoire
- Les instructions sont organisées séquentiellement
- La machine se programme par *effets de bords* (= modifications de l'état général de la machine (mémoire, vidéo, etc.))

Exemple

```
unsigned long int fact(unsigned int n) {
  unsigned long int r=1UL;
  for (; n>0u; --n)
    r *= n;
  return r;
}
```

Un espace mémoire

Un autre

A.HARICHE a.hariche@univ-dbkm.dz

Un programme impératif Exemple

- On modifie des *variables* par des *instructions*
- Les variables sont des espaces mémoire
- Les instructions sont organisées séquentiellement
- La machine se programme par *effets de bords* (= modifications de l'état général de la machine (mémoire, vidéo, etc.))

Exemple

```
unsigned long int fact(unsigned int n) {
  unsigned long int r=1UL;
  for (; n>0u; --n)
    r *= n;
  return r;
}
```

Un espace mémoire

Un autre

L'espace *r* est modifié séquentiellement dans une boucle

A.HARICHE a.hariche@univ-dbkm.dz

Un programme impératif Données et instructions

- Un programme est composé d'**instructions** qui travaillent sur des **données**.
- En programmation impérative, les données sont stockées dans des **variables**.
- Dans de nombreux langages de programmation (C/C++/Java...), il faut **déclarer** les variables.

```
#include<iostream>
void main(void) {
  double a,b;                    // Déclarations
  a=1; b=1;
  while ((((a+1)-a)-1)==0) a*=2;
  while ((((a+b)-a)-b)!=0) b++;
  std::cout << "a=" << a << ", b=", b << std::endl;
}
```


A.HARICHE a.hariche@univ-dbkm.dz

Un programme impératif
Structure de la machine

Retenir

Une machine est composée essentiellement de 4 éléments :

- *de la **mémoire** pour stocker les données*
- *des unités de **calcul** (logique, arithmétique, ...)*
- *une unité de **commande** qui organise le travail*
- *des unités de **communications** (clavier, écran, réseau, ...)*

A.HARICHE a.hariche@univ-dbkm.dz

Un programme impératif
La mémoire

La mémoire principale d'un ordinateur est composée de

- un très grand nombre (10^{10}–10^{11}) de **condensateurs** pouvant être soit chargés soit déchargés (0 ou 1)
- un gigantesque **réseau de fils et d'aiguillages** (multiplexeur)

Retenir

*On groupe les condensateurs par groupes (typiquement 8, 16, 32 ou 64) appelés **mots**. On repère un mot en mémoire grâce à un nombre appelé **adresse**.*

L'opérateur &, retourne l'adresse d'une variable :

addr.cpp

```
1    int a = 5;
2    cout << a << " " << &a << endl;
```

Affiche quelque chose comme 5 0x7ffe94f3a8bc

A.HARICHE a.hariche@univ-dbkm.dz

Un programme impératif
Notion de variable

- but : stocker des informations en mémoire centrale durant l'exécution d'un programme
- on veut éviter d'avoir à manipuler directement les adresses : on manipule des **variables**
- le programmeur donne aux variables des noms de son choix (**identificateur**)

Définition (Notion de Variable)

*Une **variable** est un espace de stockage où le programme peut mémoriser une donnée.*

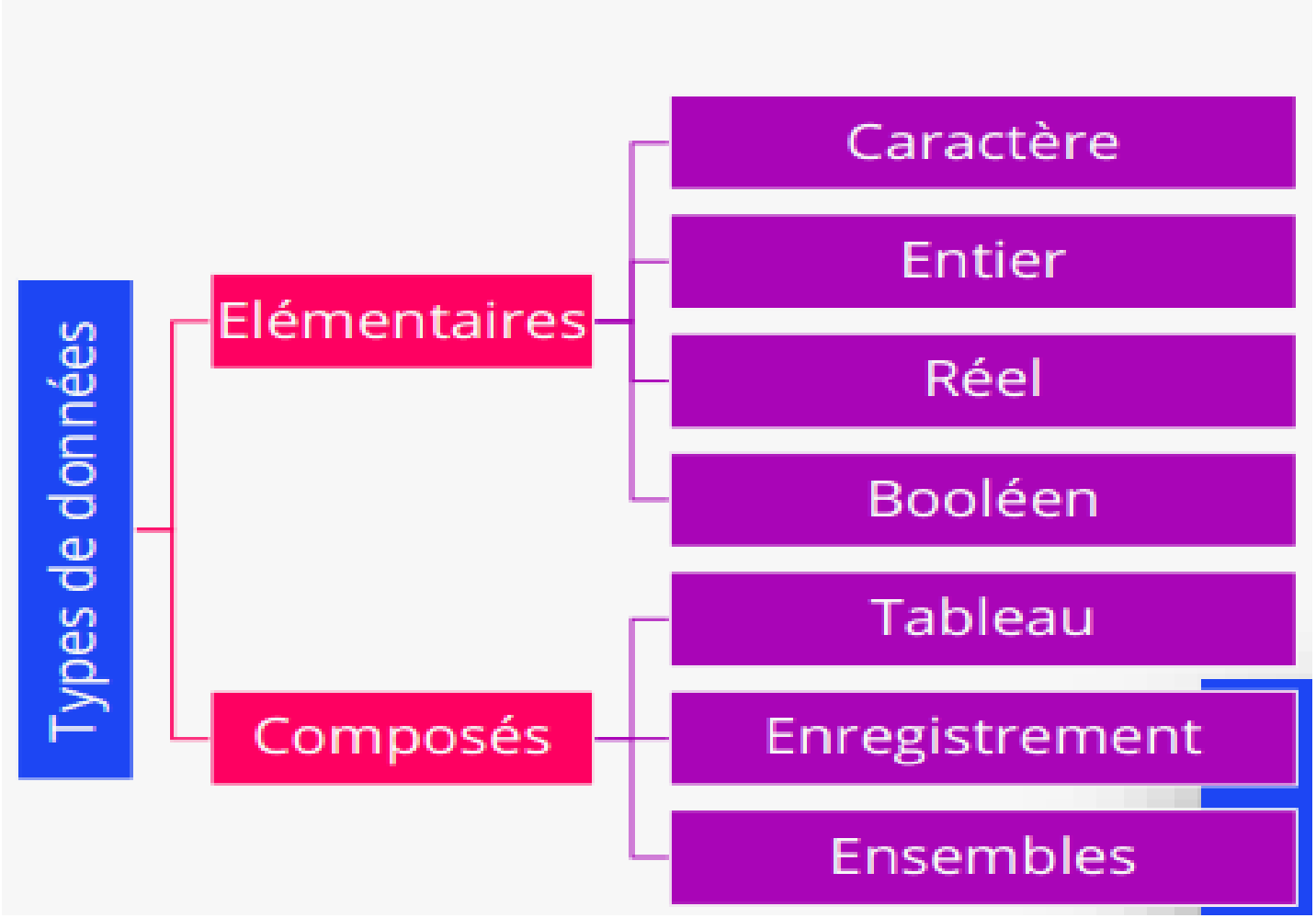
Les variables désignent une ou plusieurs cases mémoires contenant une suite de 0 et de 1.

A.HARICHE a.hariche@univ-dbkm.dz

Un programme impératif
Notion de variable

1.Nom	• sert à identifier la variable et à s'y référer. • Se rapprocher du problème décrit par le biais du langage.
2.Emplacement	• s'exprime par une adresse où toute entité peut la localisée • L'adresse est créée lorsque l'entité est créée.
3.Valeur	• c'est le contenu de la cellule mémoire associée à la variable. • le type permet d'interpréter ce contenu et de déduire la valeur.
4.Portée	• l'étendue de l'énoncé ou la variable est connue. • l'ensemble des endroits où référence à la variable peut exister • Une variable est visible dans sa portée et invisible à l'extérieur.
5.Durée de vie	• l'intervalle de temps pour lequel un emplacement mémoire de la variable existe
6.Type	• le type d'une variable est l'abstraction qui définit l'ensemble des valeurs et des opérations associées à cette variable.

Un programme impératif
Notion de variable



Un programme impératif Déclaration des variables

- Une variable peut être liée à un emplacement physique à divers moment:
- compilation (rarement),
- chargement (allocation statique),
- exécution (allocation dynamique);
- Du point de vue implémentation, la déclaration d'une variable est utilisée pour déterminer la quantité d'espace mémoire exigée par le programme ;

```
PASCAl: var name : type;
C: Type name;
```


A.HARICHE a.hariche@univ-dbkm.dz

Un programme impératif Assignation

- Une variable et une valeur sont liées par un assignement:

Exemple : V:= E:

« assigner le nom V a la valeur de l'expression E jusqu'à ce que le nom V est réassigne à une autre valeur »

PASCAL: V:=E

C: V= E;

- L'assignement n'est pas une définition de constante:

```
X:=3;
X:=X+1;
```


A.HARICHE a.hariche@univ-dbkm.dz

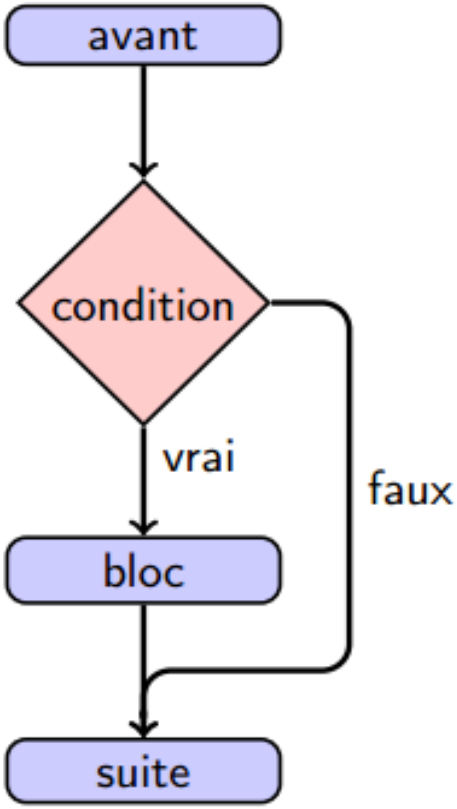
Un programme impératif
Condition

```
instructions avant;

if (condition) {
    bloc d'instructions;
}

instructions suite;
```

Remarque : s'il n'y a qu'une instruction dans un bloc les accolades ne sont pas obligatoires. On choisira ce qui est le plus **lisible**.

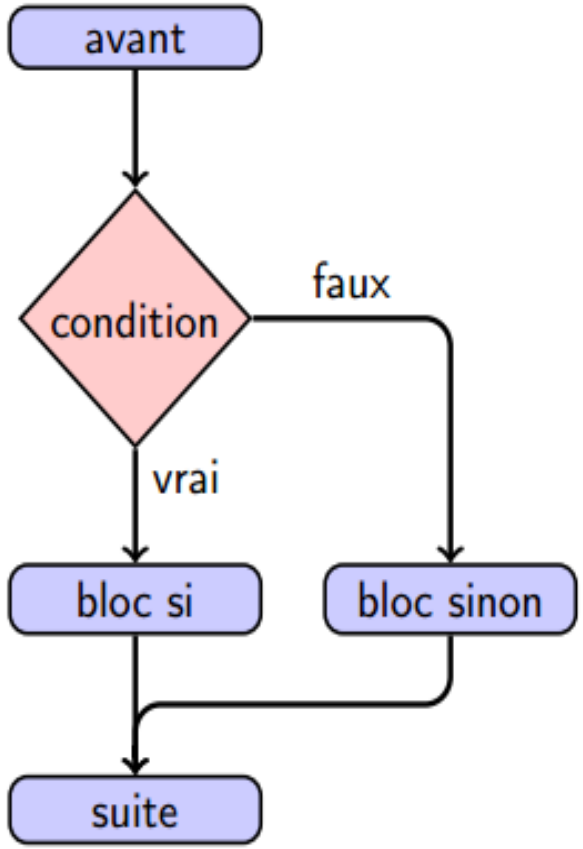


Un programme impératif Condition

```
avant;

if (condition) {
    bloc si;
} else {
    bloc sinon;
}

suite;
```



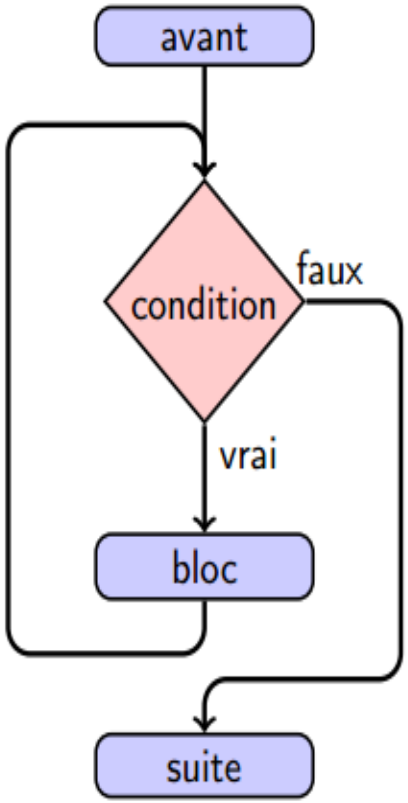
A.HARICHE a.hariche@univ-dbkm.dz

Un programme impératif Itération (boucle)-WHILE-

```
instructions avant;

while (condition) {
    bloc d'instructions;
}

instructions suite;
```




```
1  #include<iostream>
2  using namespace std;
3
4  int main() {
5    int n;
6
7    do {
8      cout << "Saisir un nombre positif :  ";
9      cin >> n;
10     if (n < 0) {
11         cout << "Ce nombre est négatif !" << endl;
12     }
13   } while (n < 0);
14   cout << "le nombre saisi est " << n << endl;
15   return EXIT_SUCCESS;
16 }
```

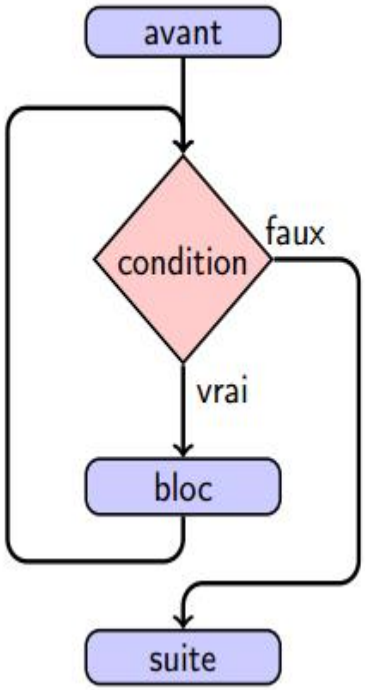
Remarque : Si la condition est fausse dès le début, le bloc n'est **jamais** exécuté.

Un programme impératif
Itération (boucle)-DO WHILE-

```
instructions avant;

while (condition) {
    bloc d'instructions;
}

instructions suite;
```



Remarque : Si la condition est fausse dès le début, le bloc n'est **jamais** exécuté.

```
1  #include<iostream>
2  using namespace std;
3
4  int main() {
5    int n;
6
7    cout << "Saisir un nombre positif :  ";
8    cin >> n;
9    while (n < 0) {
10       cout << "Ce nombre est négatif !" << endl;
11       cout << "Saisir un nombre positif :  ";
12       cin >> n;
13   }
14   cout << "le nombre saisi est " << n << endl;
15   return EXIT_SUCCESS;
16 }
```

Un programme impératif Itération (boucle)-For-

Syntaxe

```
for (initialisation; condition; incrémentation) {
     instructions;
}
```

C'est un **raccourcis plus lisible** de la boucle `while` :

```
initialisation;
while (condition) {
    instructions;
    incrémentation;
}
```


A.HARICHE a.hariche@univ-dbkm.dz

Un programme impératif
Sous programme

Syntaxe

Déclaration (mode d'emploi) :

```
type_retour nom(type1 <param1>, type2 <param2>, ...);
```

Définition :

```
type_retour nom(type1 param1, type2 param2, ...) {
  déclaration des variables locales;
  intructions;
  ...
  return valeur_de_retour;
}
```

Appel (utilisation dans une expression) :

```
   ... nom(valeur_param1, valeur_param2, ...) ...
```


A.HARICHE a.hariche@univ-dbkm.dz

Un programme impératif
Sous programme -Exemple-

Syntaxe

```
int somme(int, int);

int somme(int a, int b) {
  int res;
  res = a + b;
  return res;
}
```

```
int main() {
  int x, y, s;
  cin >> x >> y;
  s = somme (x, y);
  cout << s << endl;
  return EXIT_SUCCESS;
}
```

Un programme impératif
Sous programme -Return-

Retenir

Deux rôles :

- ***quitter** la fonction en cours ;*
- ***renvoyer** une valeur au programme appelant.*

- Dans une procédure (valeur de retour `void`), seul le premier rôle est utilisé.
- Il y a toujours une instruction `return` implicite à la fin d'une procédure.

A.HARICHE a.hariche@univ-dbkm.dz

Un programme impératif
Sous programme -Return-

Retenir

Deux rôles :

- ***quitter** la fonction en cours ;*
- ***renvoyer** une valeur au programme appelant.*

- Dans une procédure (valeur de retour `void`), seul le premier rôle est utilisé.
- Il y a toujours une instruction `return` implicite à la fin d'une procédure.

A.HARICHE a.hariche@univ-dbkm.dz

Un programme impératif
Sous programme -Récursion-

- Exercice: écrire en utilisant les sous programmes $n!$

A.HARICHE a.hariche@univ-dbkm.dz

Un programme impératif
Sous programme -Récursion-

- Exercice: écrire en utilisant les sous programmes $n!$

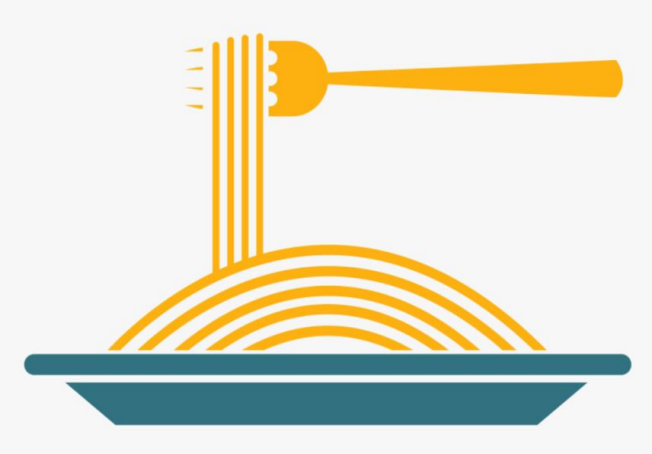
- La solution:

```
int factorielle(int n) {
    if ( n == 0 ) {
        return 1;
    } else {
        return n * factorielle(n-1);
    }
}
```


A.HARICHE a.hariche@univ-dbkm.dz

Un programme impératif Branchement

- la commande GOTO est la plus puissante, et la plus Critiquée principalement pour deux raisons :
- l'utilisation de cette instruction peut amener votre code à être plus difficilement lisible et, dans les pires cas, en faire un code spaghetti. (Cas de problème sur le système logicielle de Toyota camry 2005 en 2007).



Un programme impératif Branchement

- la commande GOTO est la plus puissante, et la plus Critiquée principalement pour deux raisons :
- mis à part dans des cas spécifiques, il est possible de réaliser la même action de manière plus claire à l'aide de structures de contrôles ;
- l'utilisation de cette instruction peut amener votre code à être plus difficilement lisible et, dans les pires cas, en faire un code spaghetti. (Cas de problème sur le système logicielle de Toyota camry 2005 en 2007).

À vrai dire, il est déconseillé de l'utiliser.

Un programme impératif Branchement

- Exercice : Voici un problème de spaghetti

Trouvez une solution à l'aide de structures itératives.

```
i = 0;
 twenty: i ++;
If (i != 11) { GOTO eighty;}
 if(i = 11) { GOTO sixty;}
 GOTO twenty;
 eighty: printf(" %d",i ," au carré = %d\n", i * i);
GOTO twenty;
sixty:printf(" Programme terminé.");
```

Un programme impératif Branchement

- Solution : une boucle for résoudra le problème.

```
for (i = 1; i <= 10; i++)  printf(" %d”,i ," au carré = %d\n“, i * i);
printf(" Programme terminé.“);
```


A.HARICHE a.hariche@univ-dbkm.dz

Paradigme impératif
Raisonnement difficile

- Certains axiomes sont valables seulement si le langage ne comporte pas de synonymie et d'effets de bord;
- Travaux de la théorie des Langages tentent d'expliquer les L. P en termes de constructions bien définies;
- Cette complexité de raisonner a été une motivation forte pour fournir des solutions comme les paradigmes fonctionnels et logiques.

Paradigme impératif
Echappement

- C'est une commande qui termine l'exécution d'une construction textuellement englobante .
  - Return Exp,

est utilisée en C pour quitter un appel de fonction et pour renvoyer la valeur calculée par la fonction.

  - Exit en ADA/ Break en C,

permet de transférer le contrôle du programme de l'endroit ou la commande exit se trouve vers la première commande après la boucle la plus interne qui suit le exit

  - Continue en C,

transfère la commande au début de la boucle englobante.

Paradigme impératif
Exceptions

- C’est un événement "anormal" qui survient au cours de l'exécution

Exercice: comment faire pour réaliser un programme calculant la fonction $\sqrt{x}$ en C résout l’exception suivante de la fonction *pow()* :

L’appel revient à demander le résultat de l’expression $-1^{\frac{1}{2}}$, autrement dit, de cette expression : $\sqrt{-1}$, ce qui est impossible dans l’ensemble des réels.

A.HARICHE a.hariche@univ-dbkm.dz

Paradigme impératif
Exceptions

Solution: nous pouvons utiliser la bibliothèque *errno* :

```
#include <errno.h>
#include <math.h>
#include <stdio.h>


int main(void)
{
   errno = 0;
   double x = pow(-1, 0.5);

   if (errno == 0)
      printf("x = %f\n", x);

   return 0;
}
```

Ou *errno* peut avoir le valeurs :
EDOM (pour le cas où le résultat d'une fonction mathématique est impossible),
ERANGE (en cas de dépassement de capacité, nous y reviendrons plus tard)
EILSEQ (pour les erreurs de conversions, nous en reparlerons plus tard également).
-Ces deux constantes sont définies dans l'en-tête <errno.h>.

Paradigme impératif
Effet de bord

- Est utilisé pour permettre la communication entre unités de programme de façon:
  - La fonction effectue modification d'E/S (1)
  - Un variable globale est modifiée (2)
  - Un variable permanente locale est modifiée (3)
  - Un paramètre de passage par référence est modifié (4)
  - Une modification est apportée aux variables non locales/statiques via des pointeurs (5)
  - Toute fonction appelée satisfait également une de ces conditions (6)

A.HARICHE a.hariche@univ-dbkm.dz

Paradigme impératif
Effet de bord

Exercice: Etudier le program suivant en terme de l'effet de bord pour les fonctions f1, f2, f3:

```
program demo;
var b : integer;
function f3 : integer;
var x : integer;
begin
x := f1 (x);
x := f2
end;
function f2 : integer;
var x : integer;
begin
x : = 20;
f2 := x
end;
```

```
function f1 (var a : integer) : integer;
begin
read (a);
a := f2;
a := f3
end;
begin
write (f1(b));
end.
```


A.HARICHE a.hariche@univ-dbkm.dz

Paradigme impératif
Effet de bord

Solution: f1 a des effets de bords vue que les conditions 1 et 4 sont franchis.

Function	Nonlocal	Ref	I/O	Indirect	Side effect
f1	Ø	[a]	[R]	Ø	true
f2	Ø	Ø	Ø	Ø	?
f3	Ø	Ø	Ø	Ø	?

Paradigme impératif
Synonymie (Aliasing)

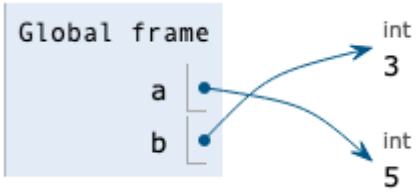
- Deux noms sont des alias s'ils dénotent (partagent) le même objet de données pendant un lancement d'unité.
- La synonymie est un autre dispositif des L. P impératifs qui rend des programmes plus durs à comprendre.

Exemple 1

```
procedure confuse (var m, n : Integer );
 begin
 n := 1; n := m + n
 end;
 ...
 i := 5;
 confuse(i,i)
```

Exemple 2

```
a = 3
b = a
a = a + 2
```



m et n sont liés à la même variable i, initialisé à 5; après l'appel la valeur de i est 2 et non pas 6.

A.HARICHE a.hariche@univ-dbkm.dz

Paradigme impératif en Bref

- La programmation impérative est basée sur le modèle machine de Von Neumann..
- Dans le paradigme impératif est caractérisé par la programmation avec un état et des commandes qui modifient l'état.
- Les difficultés liés aux paradigmes impératif peuvent être résumé comme suit
  - Les effets de bord
  - Les synonymies (aliasing)
  - Les exception

- La complexité de raisonner provenant des difficultés de la programmation impérative a été une motivation forte pour fournir des solutions comme les paradigmes fonctionnels et logiques.

A.HARICHE a.hariche@univ-dbkm.dz

Idées à apprendre

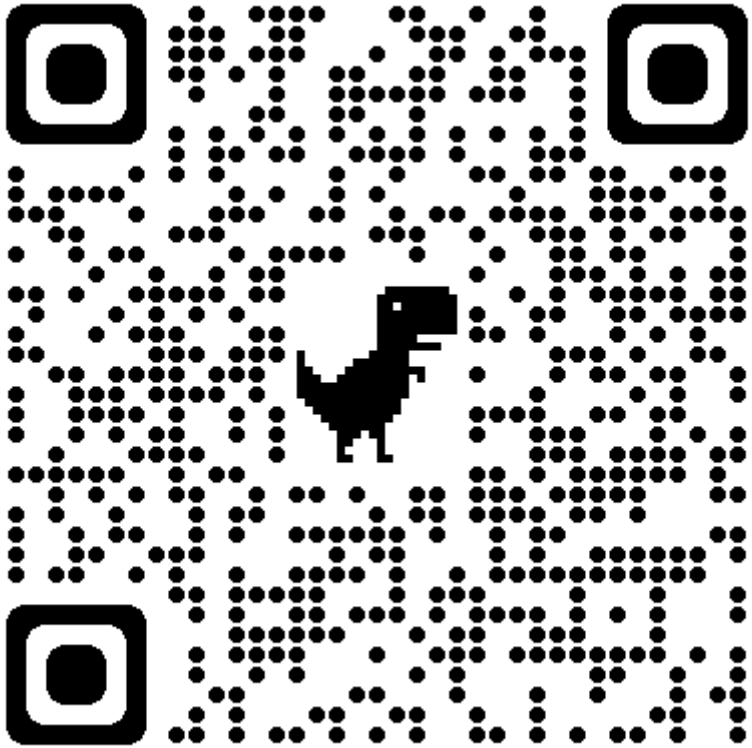
Louv1.1x

- Identifiants et environnements
- Programmation fonctionnelle
- Récursivité
- Programmation invariante
- Listes, arborescences et enregistrements
- Programmation symbolique
- Instanciation
- Généricité
- Programmation d'ordre supérieur
- Complexité et notation Big-O
- Loi de Moore
- Problèmes NP et NP-complets
- Langages du noyau
- Machines abstraites
- Sémantique mathématique

Louv1.2x

- État explicite
- Abstraction des données
- Types de données abstraites et objets
- Polymorphisme
- Héritage
- Héritage multiple
- Programmation orientée objet
- Gestion des exceptions
- Concurrence
- Non-déterminisme
- Ordonnancement et équité
- Synchronisation des flux de données
- Flux de données déterministe
- Agents et flux
- Programmation multi-agents

Lien



https://drive.google.com/drive/folders/1YBCIZzAldeiT19DIfDiREQwP-NAQ1qMN