

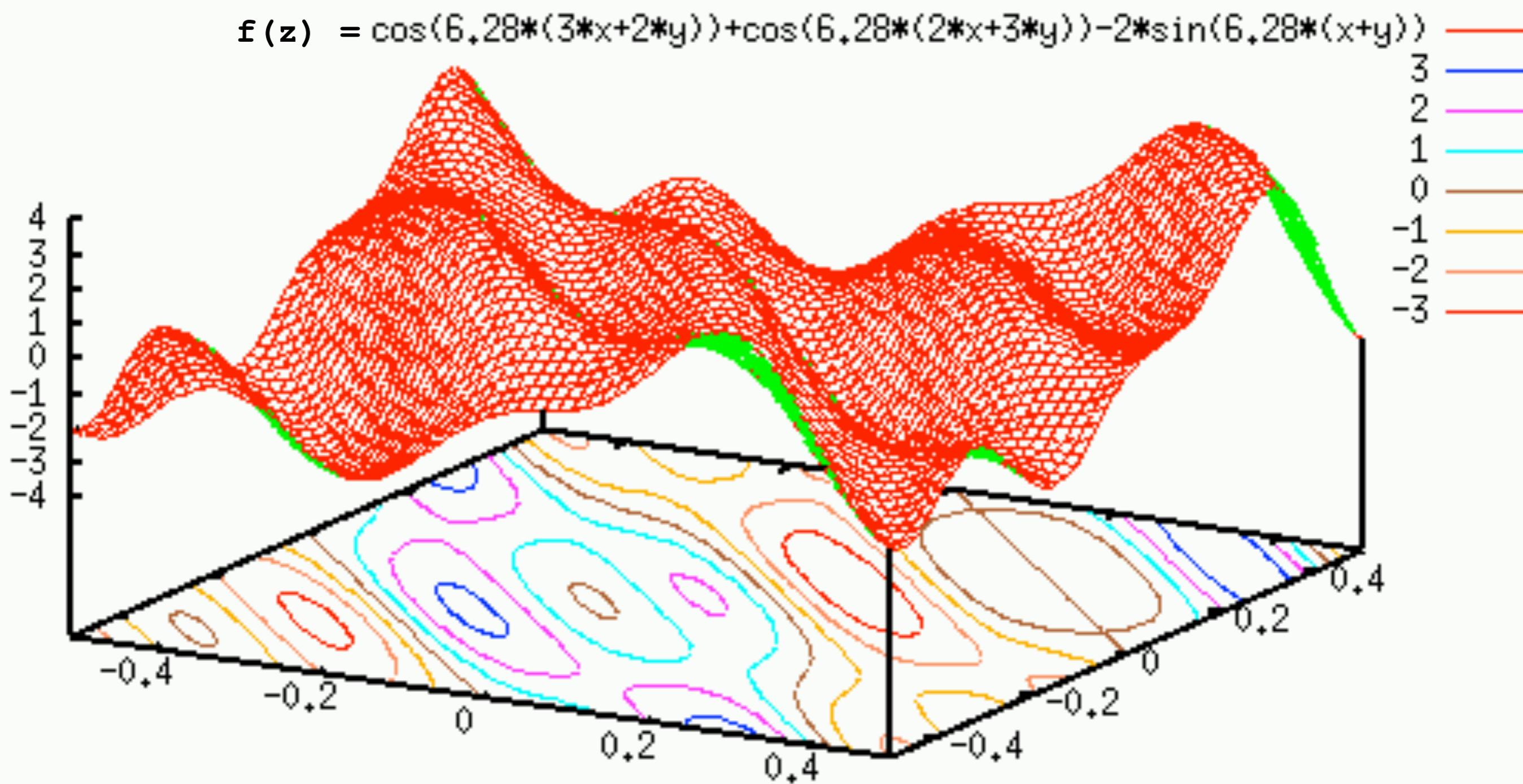
Algorithmique & Programmation

La notion de fonction

yann.secq@univ-lille.fr

ABDELKADER Omar, BIRLOUEZ Martin, BONEVA Iovka, DELECROIX Fabien, LEQUINIOU Erwann, MARSHALL-BRETON Christopher, REKIK Yosra, SECQ Yann, SOW Younoussa, SUDHEENDRAN Megha, SUE Yue

$$f(z) = \cos(6.28*(3*x+2*y)) + \cos(6.28*(2*x+3*y)) - 2*\sin(6.28*(x+y))$$



SOFT DRINK / NEWSPAPER

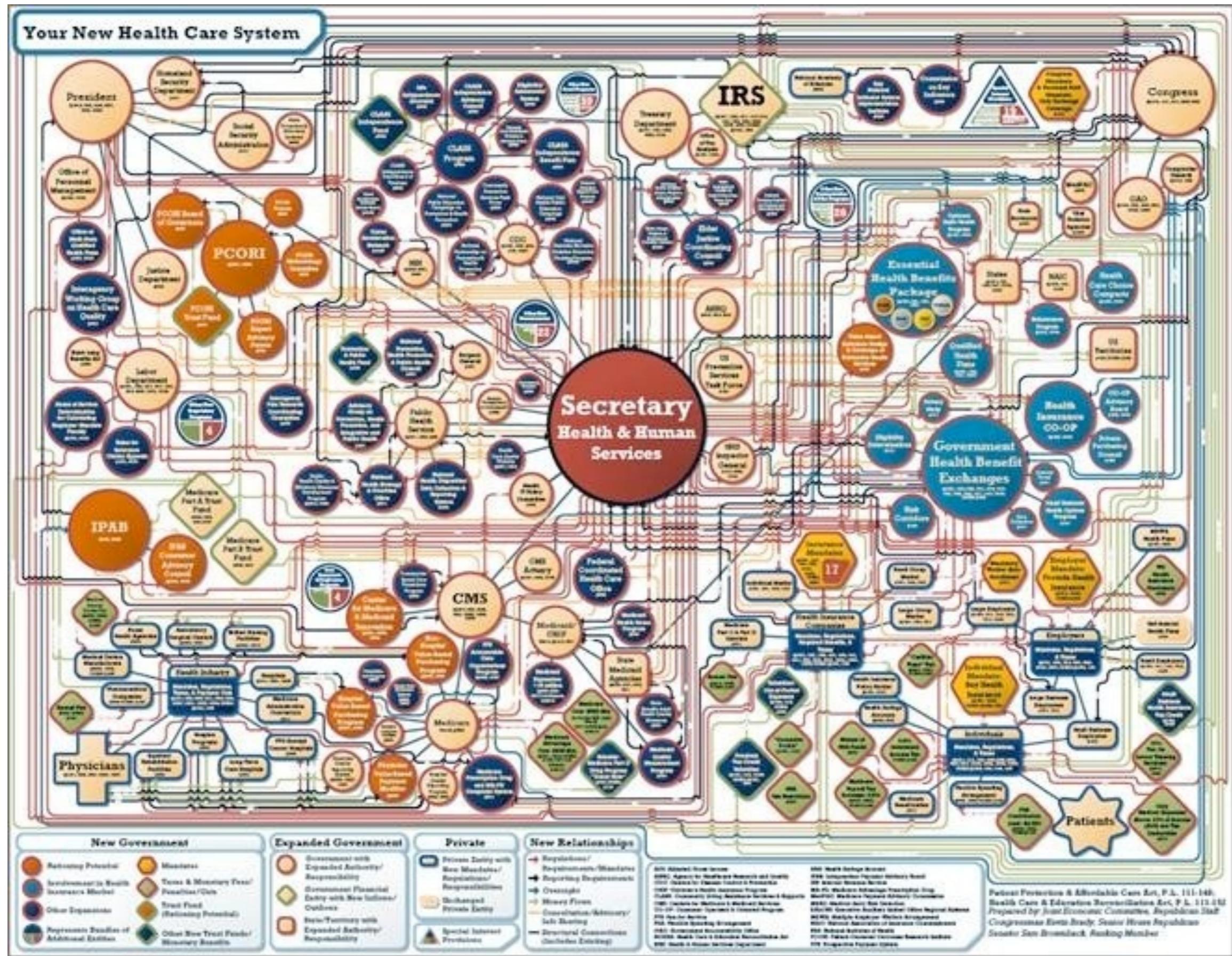
A little rest is offered to reaching your destination



La notion de fonction

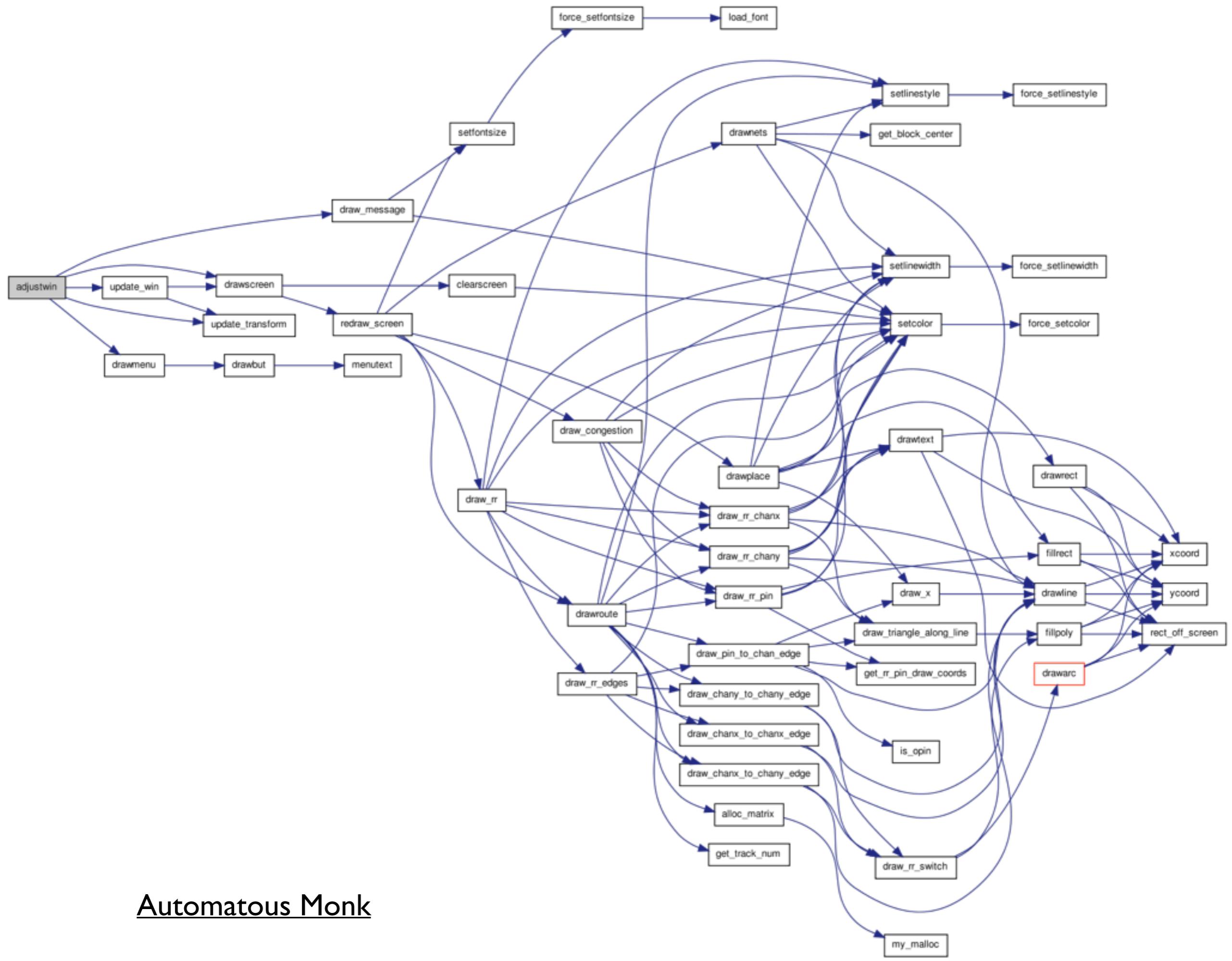
- Algorithme = [donnée(s) en entrée] + instructions + [donnée(s) en sortie]
- Fonction = algorithme réutilisable
- La notion de fonction permet de décomposer un algorithme complexe en un ensemble de sous-systèmes

Maîtriser la complexité = diviser pour régner !



Essence de la programmation ?

- Créer un langage propre au domaine modélisé
- Décomposition de systèmes complexes
- Identification de traitements récurrents
- Découpage d'un système en sous-systèmes
- Factorisation de code redondant
- Algorithme complexe = fugue d'appels de fonctions



Automatous Monk

Complexité (et intérêt !) de la programmation

- Comment structurer des programmes ?
- Comment faciliter leur création ?
- Comment améliorer leur qualité ?
- Comment réutiliser des algorithmes ?

Un exemple simple

```
class CorrigerTexte extends Program {  
  
    // définition de la fonction copieSans ...  
  
    void algorithm() {  
        String avant = "La disparition";  
        String apres = copieSans(avant, 'i');  
        println(apres); // La dsparton  
    }  
}
```

Un exemple simple

```
class CorrigerTexte extends Program {  
  
    // définition de la fonction efface ...  
  
    void algorithm() {  
        String avant = "La disparition »;  
        String apres = copieSans(avant, 'i');  
        println(apres); // La dparton }  
    }  
}
```

*Appel d'une fonction nommée
println avec une valeur
(une chaîne de caractères)*

*Appel d'une fonction
nommée copieSans
avec deux valeurs (une
chaîne et un caractère)*

Un exemple simple

```
class CorrigerTexte extends Program {  
  
    // définition de la fonction efface ...  
  
    void algorithm() {  
        String avant = "La disparition »;  
        String apres = copieSans(avant, 'i');  
        println(apres); // La dparton  
    }  
}
```

Fonction
prédéfinie

Fonction
à définir !

Analyse de l'appel de fonction

```
String avant = "La disparition";  
String apres = copieSans(avant, 'i');
```

Type du résultat produit

Nom de la fonction

Données nécessaires pour réaliser le traitement (ou paramètres)

```
(avant, 'i')
```

Une chaîne de caractères String

Un caractère char

```
String copieSans(String phrase, char lettre)
```

Un exemple simple

```
class CorrigerTexte extends Program {  
  
    // Signature de la fonction  
    String copieSans(String phrase, char lettre) {  
        // corps de la fonction  
    }  
  
    void algorithm() {  
        String avant = "La disparition »;  
        String apres = copieSans(avant, 'i');  
        println(apres); // La dsparton  
    }  
}
```

```
class CorrigerTexte extends Program {  
  
    String copieSans(String msg, char c) {  
        String resultat = "";  
        for (int i=0; i<length(msg); i=i+1) {  
            if (charAt(msg, i) != c) {  
                resultat = resultat + charAt(msg, i);  
            }  
        }  
        return resultat;  
    }  
  
    void algorithm() {  
        String avant = "La disparition »;  
        String apres = copieSans(avant, 'i');  
        println(apres); // La dsparton  
    }  
}
```

Définition de la fonction copieSans

Appel de la fonction copieSans

Une fonction peut appeler une autre fonction !

```
class CorrigerTexte extends Program {  
  
    String copieSans(String phrase, char lettre) {...}  
  
    String copieSansVoyelles(String phrase) {  
        String resultat = copieSans(phrase, 'a');  
        resultat = copieSans(resultat, 'e');  
        resultat = copieSans(resultat, 'i');  
        resultat = copieSans(resultat, 'o');  
        resultat = copieSans(resultat, 'u');  
        return resultat;  
    }  
  
    void algorithm() {  
        String avant = "La disparition";  
        String apres = copieSansVoyelles(avant);  
        println(apres); // L dsprtn  
    }  
}
```

La notion de fonction

- **Une fonction est un algorithme réutilisable**
- Une fonction nécessite des **informations en entrée** et produit **un résultat en sortie**
- Une fonction peut utiliser une autre fonction (ou elle même, mais on verra cela plus tard ...)
- Instructions que l'on connaît = fonctions prédéfinies !

Notion de portée

- Les variables ont une portée bien définie
- **Une variable n'existe que dans le bloc où elle est déclarée**
- Avant, la vie était simple ... (sauf for !)
- Maintenant, soyez attentifs à cette notion de portée avec les fonctions !

```
class CorrigerTexte extends Program {  
    String nom = "Turing";  
    String copieSans(String msg, char c) {  
        String resultat = "";  
        for (int cpt=0; cpt<length(msg); cpt=cpt+1) {  
            if (charAt(msg, cpt) != c) {  
                resultat = resultat + charAt(msg, cpt);  
            }  
        }  
        return resultat;  
    }  
  
    void algorithm() {  
        String texte;  
        texte = readString();  
        println(copieSans(texte, 'e'));  
        println(copieSans(nom, 'u'));  
    }  
}
```

Et si resultat était nommée texte ?

Et si nom était nommée texte ?

nom est une variable **globale**

resultat est une variable **locale** à copieSans

i est une variable **locale** à la boucle **for**

texte est une variable **locale** à algorithme

```
class CorrigerTexte extends Program {  
    String nom = "Turing";  
    String copieSans(String msg, char c) {  
        String resultat = "";  
        for (int i=0; i<length(msg); i=i+1) {  
            if (charAt(msg, i) != c) {  
                resultat = resultat + charAt(msg, i);  
            }  
        }  
        return resultat;  
    }  
    void algorithme() {  
        String texte;  
        texte = readString();  
        println(copieSans(texte, 'e'));  
        println(copieSans(nom, 'u'));  
    }  
}
```

Synthèse

- **Une fonction est définie par sa signature :**
 - le **nom** de la fonction (le plus pertinent possible !)
 - les **paramètres** (informations nécessaires pour réaliser le calcul)
 - le **type de son résultat**
- **ATTENTION : un seul return par fonction et toujours en dernière instruction !**
- Importance de la **notion de portée des variables** et paramètres

Synthèse : appel et signature de fonction

```
String avant = "La disparition";  
String apres = copieSans(avant, 'i');
```

APPEL

Type du résultat produit

Nom de la fonction

Données nécessaires pour réaliser le traitement (ou paramètres)

Une chaîne de caractères
String

(avant, 'i')

Un caractère
char

```
String copieSans(String phrase, char lettre)
```

SIGNATURE DE FONCTION

Nombre d'occurrences

```
class NbOccurrences extends Program {  
  
    void algorithm() {  
        String phrase = "La disparition";  
        char lettre = 'i';  
        int nbOccurrences = 0;  
        for (int idx=0; idx < length(phrase); idx=idx+1) {  
            if (charAt(phrase, idx) == lettre) {  
                nbOccurrences = nbOccurrences + 1;  
            }  
        }  
        println("Il y a "+nbOccurrences+" fois la lettre "  
              +lettre+" dans \"\""+phrase+"\");  
    }  
}
```

Déplacement du traitement dans une fonction !

Nombre d'occurrences

```
class NbOccurrences extends Program {  
  
    void algorithm() {  
        String phrase = "La disparition";  
        char lettre = 'i';  
        int nbOccurrences = 0;  
        for (int idx=0; idx < length(phrase); idx=idx+1) {  
            if (charAt(phrase, idx) == lettre) {  
                nbOccurrences = nbOccurrences + 1;  
            }  
        }  
        println("Il y a "+nbOccurrences+" fois la lettre "  
              +lettre+" dans \""+phrase+"\");  
    }  
}
```

Déplacement du traitement dans une fonction !

Nombre d'occurrences

```
class NbOccurrences extends Program {
    int nombreOccurrences(String mot, char lettre) {
        int nbOccurrences = 0;
        for (int idx = 0; idx < length(mot); idx=idx+1) {
            if (charAt(mot, idx) == lettre) {
                nbOccurrences = nbOccurrences + 1;
            }
        }
        return nbOccurrences;
    }
    void algorithm() {
        String phrase = "La disparition";
        char symbole = 'i';
        println("Il y a "+
               nombreOccurrences(phrase, symbole) +
               "fois la lettre "+symbole+" dans \"\""+phrase+"\"");
    }
}
```

Un random plus pratique

```
class RandomPratique extends Program {  
  
void algorithm() {  
    int min = 1;  
    int max = 6;  
    int alea = min + (int) (random()*(max-min+1));  
    println(min+" <= "+alea+" <= "+max);  
}  
  
}
```

Un random plus pratique

```
class RandomPratique extends Program {  
  
    int random(int borneMin, int borneMax) {  
        int alea = borneMin +  
                  (int) (random() * (borneMax-borneMin+1)) ;  
        return alea;  
    }  
    void algorithm() {  
        int min = 1;  
        int max = 6;  
        println(min+" <= "+random(min, max)+" <= "+max);  
    }  
}
```

Déplacement du traitement dans une fonction !

Le jeu de Nim

- Jeu à information complète à 2 joueurs
- N allumettes, chacun en tire entre 1 et 3 par tour, celui prenant la dernière perd
- Comment décomposer la description de ce jeu pour faciliter sa programmation ?

Le jeu de Nim : analyse

- Jeu à information complète à 2 joueurs
- n allumettes, chacun en tire entre 1 et 3 par tour, celui prenant la dernière perd

● Quelles données ?

- Allumettes ? Joueurs ?
- Quels usages ? Quels types ? Variation ou pas ?

● Quels traitements ?

- Notion de tour de jeu
- Notion de fin de gain d'une partie

