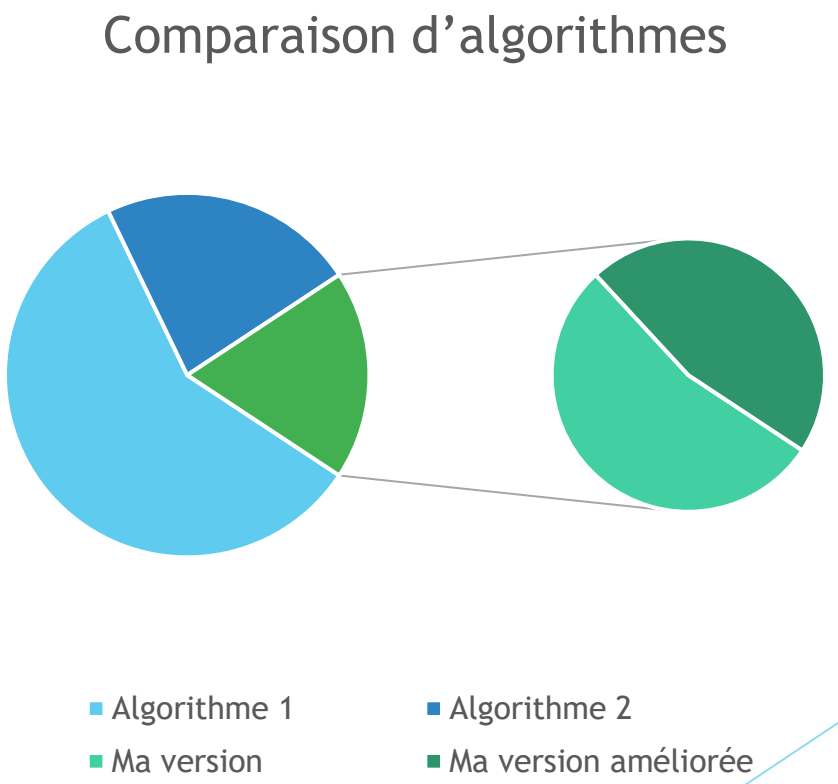
Document de présentation TIPE

Quelques trucs et astuces

Une idée générale par diapo

- On aborde un seul thème/une seule notion à présenter
- On évite de trop rentrer dans les détails d'une preuve

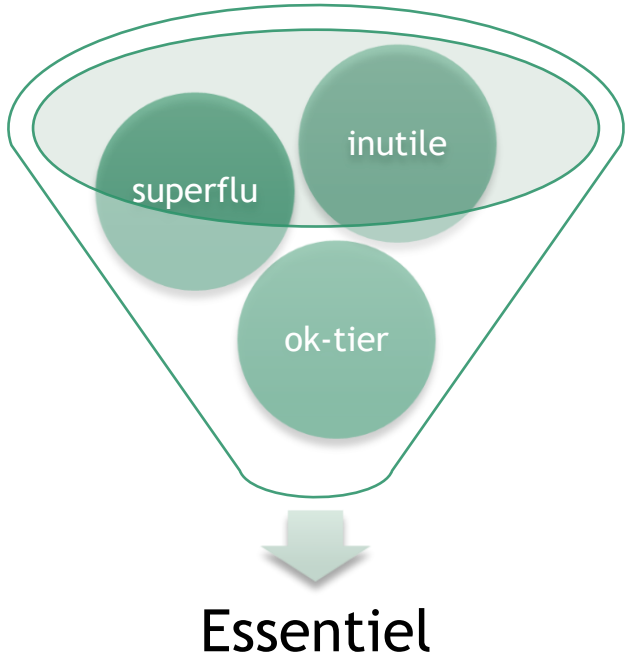


Éviter de faire des diapos qui contiennent trop d'informations sinon on s'y perd dans le détail, et après on s'ennuie un peu… Surtout, évitez de faire des titres trop longs qui débordent

- On fait attention que le texte doit bien être lisible
- On évite de se répéter plusieurs fois sur la diapo
- Chaque diapo doit contenir des informations intéressantes, qui ne sont pas répétées
- S'il y a trop d'information, les choses importantes sont noyées dans le reste
- Les bullets-points, ça va pour 2-3 diapos, mais faut faire attention à ne pas mettre que ça, sinon on s'ennuie vite
- Faut éviter de dire la même chose à plusieurs endroits sur la diapo, sinon c'est répétitif
- Il faut limiter le nombre de figures également, 2 figures maximum par diapo
- Le texte doit être assez gros : on doit pouvoir le lire depuis le milieu de la classe

Prévoir une minute par diapo !

- Ça veut dire de l'ordre de ~15 diapos au total !
- Il faut faire le tri et aller à l'essentiel !



5 Ici

Ici 5

On numérote les diapos

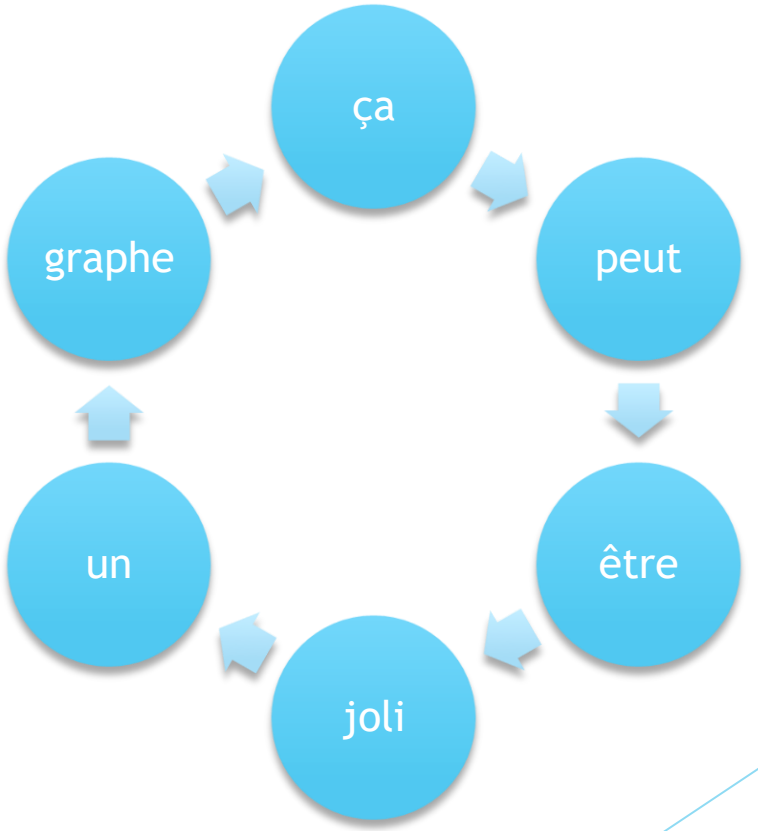
5 Ici

ou ici 5

Esthétique sobre, mais pas nécessairement absente

- Les fonds tout blancs, ça passe, mais c'est un peu triste

- On peut mettre des diagrammes, même si ça fait un peu « pipeau » (attention à rester raisonnable)



Attention aux choix des couleurs

- Pas de couleurs vives
- On fait attention au contraste entre texte et fond
- On projette avec un vidéoprojecteur pour tester

Pas de Comic Sans

Non, sérieusement, vous êtes à une présentation de concours, pas dans une carte d'invitation à votre anniversaire.

Pas plus de 3 lignes de code !

```
let rec p_egal_np x =
    let y = solveur_sat_polynomial (phi x) in
    clique y
```

Si on veut mettre du code, on cible les idées essentielles.

Non, sérieusement, pas plus de 3 lignes de code.

```
let maxsat_bnb phi =
    (* On écrit alors l'algorithme de Branch & Bound avec les deux heuristiques. *)
    let taille_max = ref 0 and
        mu_max = ref [||] in
    let n = taille_V phi in
    let mu = Array.make n 0 in
    let rec bnb i =
        if i = n then begin
           let h = taille_J phi mu in
           if h > !taille_max then begin
              taille_max := h;
              mu_max := Array.copy mu
           end
        end else begin
           let h = heuristique phi mu i in
           if h > !taille_max then begin
              let b = branchement phi mu i in
              mu.(i) <- b;
              bnb (i + 1);
              mu.(i) <- 1 -
              bnb (i + 1)
           end
        end in
    bnb 0;
    !mu_max;;
```

```
def convergence_newton(racines, z0, e, itermax):
    z1 = z0 - P(racines, z0) / Pprime(racines, z0)
    k = 0
    while abs(z1 - z0) > e and k < itermax:
        k += 1
        z0, z1 = z1, z1 - P(racines, z1) / Pprime(racines, z1)
    for i in range(len(racines)):
        if abs(z1 - racines[i]) <= e: return (i, k)
    return (-1, k)

rouge, vert, bleu = np.array([255, 0, 0]), np.array([0, 255, 0]), np.array([0, 0, 255])

def fractale(racines, L, H, a, b, c, d,  e, itermax):
    image = np.zeros((H + 1, L + 1))
    for j in range(H + 1):
        for k in range(L + 1):
            re = a + k * (b - a) / L
            im = c + (H - j) * (d - c) / H
            z0 = re + 1j * im
            i, l = convergence_newton(racines, z0, e, itermax)
            image[j, k] = i
    return image
```

```
bool CYK(grammaire G, char* u){
    int n = strlen(u);
    if (n == 0){
        for (int p=0; p<G.nb_prod; p++){
            if (G.Prod[p].X == 'A' && strlen(G.Prod[p].alpha) == 0){
                return true;
            }
        }
        return false;
    }
    bool*** tab_X = init_tab_X(n, G.taille_V);
    for (int i=0; i<n; i++){
        for (int p=0; p<G.nb_prod; p++){
            int t = strlen(G.Prod[p].alpha);
            if (t == 1 && G.Prod[p].alpha[0] == u[i]){
                tab_X[i][i][indice(G.Prod[p].X)] = true;
            }
        }
    }
    for (int d=1; d<n; d++){
        for (int i=0; i<n-d; i++){
            int j = i + d;
            for (int k = i; k<j; k++){
                for (int p=0; p<G.nb_prod; p++){
                    int t = strlen(G.Prod[p].alpha);
                    if (t == 2){
                        char Y = G.Prod[p].alpha[0];
                        char Z = G.Prod[p].alpha[1];
                        if (tab_X[i][k][indice(Y)] && tab_X[k+1][j][indice(Z)]){
                            tab_X[i][j][indice(G.Prod[p].X)] = true;
                        }
                    }
                }
            }
        }
    }
    bool b = tab_X[0][n-1][0];
    liberer_tab_X(tab_X, n);
    return b;
}
```

Les formules sont des formules…

- On utilise un environnement de formules (inclus ou externe)

$$(x+a)^n = \sum_{k=0}^{n} \binom{n}{k} x^k a^{n-k}$$

Les formules sont des formules…

- On fait attention si on importe des images à la netteté, au fond et aux alignements

$$(x+a)^n = \sum_{k=0}^{n} \binom{n}{k} x^k a^{n-k} \quad = 42$$

…et pas des suites moches de symboles

- On évite d'écrire sans environnement mathématique
- Sinon, c'est moche.

(x + a)^n = Σ_k=0^n bin(n,k)*x^k*a^(n-k)

Si on utilise des items

- On
- utilise
- l’environnement
- prévu
- pour.

On évite…

- Les items précédés par des tirets

- Même pas bien alignés

- Et avec un interligne aléatoire

Attention aux fôtes

Prenez le temps de vous relire

On finit par une diapo récapitulative

- Propre
- Lisible
- Clair
- Précis

Jury content

☺

Filler

Filler

Filler

Filler

Filler

Filler

Filler

Filler

Filler

Filler

Filler

Filler

Filler

Filler

Filler

Filler

Filler

Filler

Filler

Filler

Filler

Filler

Filler

Filler

Filler

Filler

Filler

Filler

Filler

Filler

Filler

Filler

Filler

Filler

Filler

Filler