- Attention: pas de zip, pas de sous dossier (Maxence, Timeo)
- Rendre à l'heure. Plus de tolérance
- Attention aux noms des fichiers, aux formats (pas de HEIC!!)
- Renommez les scan et les images SVP avant de les envoyer NOM_partie_papier.jpg par exemple
- Attention à ne pas me mettre les fichiers temporaires qui se terminent par ~…
- Regardez mes codes corrigés que je vous mets à disposition
- Respectez bien les spécifications!!! Si on vous demande une fonction qui prend tant d'entrées et renvoie tant de sorties, respectez scrupuleusement
- Beaucoup ne m'ont pas rendu la partie "papier"... notée sur 4 points quand même... Accolades lourdes et inutile quand il y a une seule instruction sous un if/for/while
- Travaillez l'indentation, les alignements, les commentaires... La qualité de vos codes sera regardée par vos examinateurs
- Réindexation automatique: en sélectionnant tout et en faisant Ctrl+Alt + antislash

- Diagramme entrée/sortie d'un algo: on en est pas au stade du passage par adresse. Le passage par adresse est juste un "truc". Bien mettre en entrée ce qui constitue des entrées au niveau algorithmique.
- Afficher une valeur est différent de retourner une valeur.
- Erreurs algorithmiques : prendre le temps d'écrire le pseudo code
- Programmation défensive assert
- Quand vous voulez avez des variables vrai/faux → booléens dans les deux langags
- Il faut passer le minimum vital à la fonction pour limiter les mauvais usages de la fonction. Code Marius MISPLON, Maxime ex2
- L'utilisation d'un for au lieu d'un while, quand c'est possible, est plus sure (code Paul ex3)
- Pensez à désallouer l'espace mémoire alloué manuellement avec malloc dans le segment du tas + remettez d'adresse du pointeur à NULL apres avoir libéré la mémoire qui était pointée par ce pointeur

Exercice 1

- Attention aux spécifications: on renvoie un TABLEAU, le prototype doit donc être 'a array -> 'a array <fun>

  REGARDEZ A CHAQUE FOIS LES PROTOTYPES DE FONCTION INFERES PAR L'INTERPRETEUR OCAML

- Afficher une valeur est différent de retourner une valeur. Code Léo par exemple, ou code Maé
- Attention à l'excès de point virgule en OCaml
  Montrer le code d'Elsa
- Attention, les valeurs qui ne sont utiles que dans la fonction doivent être définies dans la fonction. Code de Maé
- L'opérateur = en OCaml n'est pas l'opération d'affectation comme en C. L'opérateur = est le test d'égalité en OCaml. En OCaml, l'opération d'affectation ne peut se faire que sur des variables modifiables, c'est à dire des références, avec l'opérateur := Code Maxime
- Erreur sur l'algo: double boucle, exemple Tristan Panazzolo
- En mode interactif, pas trop la peine de s'embêter. Si vous écrivez simplement tableau;; l'interpréteur va évaluer cette valeur et l'afficher. Les Printf.printf sont utiles surtout lorsque

l'on est pas en mode interactif, lorsqu'on lance le script en un seul coup dans la console avec > ocaml tri.ml

- Attention, les valeurs qui ne sont utiles que dans la fonction doivent être définies dans la fonction. Code de Maé

- **L'opérateur = en OCaml n'est pas l'opération d'affectation comme en C : l'opérateur = est le test d'égalité en OCaml**.

  En OCaml, l'**opération d'affectation** ne peut se faire que sur des variables modifiables, c'est à dire des références, avec l'opérateur := ou ← Code Maxime

  Test d'égalité en OCaml. Il y a deux fonctions d'égalité en OCaml: = et == qui sont toutes deux polymorphes de type: 'a -> 'a -> bool. Elles ne doivent pas être confondues car elles sont sémantiquement différentes:
  - == teste l'identité de ses arguments, en testant l'égalité physique des représentations mémoires. == est très efficace car ce prédicat correspond généralement à une instruction de la machine. == s'applique à tout couple de valeurs du même type sans provoquer d'erreurs: pour toutes les valeurs non allouées on teste l'identité bit-à-bit de leur représentation, et pour toutes les valeurs allouées on teste si elles sont stockées à la même adresse mémoire
  - = teste l'égalité sémantique: les arguments sont-ils isomorphes ? Pour cela = parcourt en parallèle les deux données jusqu'à trouver une différence, ou avoir terminé le parcours. C'est pourquoi = peut boucler en cas de valeurs cycliques.

  - En résumé: == est
    - efficace puisqu'il est implémenté à l'aide d'une seule instruction machine
    - général: on peut l'appliquer à n'importe quelle paire de valeurs, il n'échoue jamais.
    - sémantiquement significatif quand il retourne vrai, mais peut être sémantiquement
    - non significatif quand il retourne faux, puisque deux valeurs physiquement distinctes peuvent se révéler sémantiquement égales (par exemple deux chaînes de caractères comportant les mêmes caractères ne seront pas égales au sens de ==).
    - d'un emploi plus délicat que =: il faut avoir une certaine connaissance de la représentation interne des valeurs pour l'utiliser à bon escient (par exemple deux nombres flottants ``évidemment égaux'' ne sont pas == avec la plupart des compilateurs Caml).

  - En revanche, = est
    - moins efficace puisqu'il est implémenté à l'aide d'un parcours récursif de ses arguments.
    - moins général: on ne peut pas l'appliquer à n'importe quelle paire de valeurs, il échoue en particulier quand il rencontre des valeurs fonctionnelles.
    - sémantiquement significatif quand il retourne faux, mais peut être sémantiquement non significatif quand il retourne vrai, puisque deux valeurs ayant les mêmes composantes peuvent se révéler sémantiquement différentes (par exemple les valeurs qui comportent des valeurs mutables).
    - peut ne pas terminer: en cas de valeurs comportant des cycles = peut boucler à jamais (il serait sans doute concevable d'améliorer le prédicat pour qu'il utilise un algorithme qui termine plus souvent).
    - c'est cependant le prédicat d'égalité usuel: il implémente la notion d'égalité la plus intuitive et la plus raisonnable.

Exemples:
Les chaînes de caractères sont des données allouées en mémoire, la différence sémantique entre = et == est donc évidente:

```
#let s = "ok";;
s : string = "ok"
#s == s;;
- : bool = true
#s == "ok";;
- : bool = false
#s = "ok";;
- : bool = true
#"ok" == "ok";;
- : bool = false
#"ok" = "ok";;
- : bool = true
```

Le comportement sur les valeurs mutables est aussi caractéristique: == est plus juste que =, puisque = ne répond pas toujours la même chose lorsque soumis aux mêmes données.

```
#let x = ref 1 and y = ref 1;;
x : int ref = ref 1
y : int ref = ref 1
#x = y;;
- : bool = true
#x == y;;
- : bool = false
#x := 2;;
- : unit = ()
#x = y;;
- : bool = false
#x == y;;
- : bool = false
```

Pour les flottants le prédicat == n'est pas approprié:

```
1.0 == 1.0;;
- : bool = false
```

Ces deux flottants sont donc alloués. Pour les flottants il faut donc impérativement utiliser l'égalité =.

Pour les listes, le même phénomène se produit: comparer des listes à l'aide de == n'a pas grand sens:

```
#[1] == [1];;
- : bool = false
#[1] = [1];;
- : bool = true
```

Ce phénomène est généralement vrai pour toutes les structures de données non mutables: les

comparer avec le prédicat = donne les résultats les plus intuitivement évidents.

Exercice 2

- Quand utiliser les références :
  Par défaut, une variable est immuable
  Références : pour les variables modifiables, dont le contenu évolue au cours de l'algo ---> référence
- Utilisez les booléens pour dire oui/non vrai/faux plutot que 0/1
- Évaluation versus déclaration. Code de Margot

  - let toto = val est une déclaration
    nom;; est une évaluation
    let tutu = v in tutu;; est une évaluation
- Montrer utilisation du filtrage code Timéo

Exercice 3

- Pourquoi test sur le premier caractère du motif?? Autant commencer la sous-boucle à j = 0
- Non respect des spécifications: il faut retourner l'ADRESSE du premier caractère de la premiere occurence, pas son indice
  Renvoyer l'adresse de la 1ere occurrence
  ==> renvoyer dans un pointeur (return ou passage par adresse) l'adresse de la première case contenant la 1ere occurrence du motif
- Valeurs retour, il y en a 2 donc au moins l'une des deux devra être passée par adresse
  Montrer code Olivier GRELY: pourquoi ca ne marhce pas
  Montrer code elsa
  Montrer code Paolo: les deux sont passés par adresse
- Pseudo code souvent oublié
- Complexité N*M car double boucle. Un ordre de grandeur pour N et M grand suffit (comportement asymptotique). On dit que la complexité est quadratique
- Affichage %s pour les chaines de caractères (pour n'importe quel char *, même un pointeur qui pointeur sur un caractère qui est au milieu d'une autre chaîne)
- Syntaxe break pour casser une boucle en C, pas possible en OCaml

Exercice 4

- Non respect des spécifications: tableau codé de manière bidimensionnelle, montrer code Romain Loiseau
- Ne trichez pas et respectez les spécifications: il fallait définir une fonction allouer_matric
- Deux possibilités pour retourner l'adresse de la matrice allouée en sortie
  Montrer le code de Mae
- Passage par valeur dans allouer_matrice mais pas dans les autres, inutile, il faut réfléchir!

- Pas besoin de retourner la matrice dans remplir_matrice code Margot

- Pensez à désallouer la matrice
  Sécurité
  if (mat != NULL)
   free(mat)
  mat = NULL;