# Informatique 1
# Introduction à la programmation

## Questions concernant l'examen

**Kim Mens** **Siegfried Nijssen**

# Questions possibles…

*Est ce que notre code sera encore corrigé manuellement?*

*Avons nous le droit d'utiliser les slides du cours ou nos notes?*

*Pourriez-vous expliquez comment la note finale sera calculée?*

*Quelles sont les bonnes pratiques standard à appliquer dans notre création de code?*

*Est ce qu'on aura accès à nos codes des différentes missions sur inginious?*

*Peut-t-on pour l'examen écrire nos programmes sur Thonny ?*

*Faut-il utiliser la programmation défensive pour vérifier les pre- et postconditions?*

*Est-ce qu'on pourra utiliser des méthodes Python qui n'ont pas été présentées explicitement au cours ?*

*Pendant l'examen, est-ce qu'on a accès au syllabus comme sur l'examen blanc?*

*Combien de temps aura t'on pour réaliser l'examen ?*

*Doit-on mettre des commentaires, des spécifications pre-post, etc. ?*

# …concernant les modalités de l'examen

# Modalités pratiques de l'examen

- Date : jeudi 18 janvier 2024
- Durée : de 14h à 17h (3 heures)
  - On vous conseille vivement de participer activement à l'examen.
- En E6K – Salle 13 sur INGInious
- Vous ne pouvez pas utiliser Thonny à l'examen

# Introduction

# INGInious

Collapse context

La durée de l'examen est 3 heures.

Cette examen comporte 6 questions.

Chaque question est évaluée indépendamment des autres et possède son contexte.
Des tests vous sont fournis afin de vous aider à évaluer votre code. *Attention*, passer ceux-ci r
question. Veillez donc à ajouter vos propres tests.

*Attention* qu'ici seule la syntaxe de votre code est évaluée. Un pourcentage à 100% à une question indiquera donc juste un code
syntaxiquement correct. Il faudra en outre qu'il respecte les spécifications afin de passer nos

Dans aucune question il ne sera possible de faire des `import`.

## Ressources

Pendant l'examen, vous pouvez consulter les ressources suivantes:

- Syllabus interactif
- Syllabus Format PDF
- Open a new window to INGInious

## Zone de test

Voici une zone de test Python. Insérez du code de test Python dans la boîte ci-dessous. Vous pouvez utiliser `print` pour tester votre code. Ce code de test est exécuté en Python, en dehors de toute autre définition. Vous pouvez utiliser cette zone par exemple pour obtenir la documentation Python avec la fonction `help`. Exemple:

```
help(print)
```

### Runner python

Il s'agit d'un runner Python libre que vous pouvez utilisez pour exécuter du code Python. l
apparaîtront dans le feedback de retour.

```
1
```

Pendant l'examen, est-ce qu'on a accès au syllabus interactif?

OUI, au syllabus **théorie** + en PDF

*Est ce qu'on aura accès à nos codes des différentes missions sur INGInious?*

NON, pas au syllabus **d'exercices** ni à vos soumissions sur INGInious

Avons nous le droit d'utiliser les slides du cours ou nos notes?

NON, que le syllabus théorie

# Consignes de confidentialité

- Durant l'examen, vous pouvez consulter le **syllabus théorie du cours**
- Mais pas le **syllabus d'exercices**, ni vos **soumissions INGInious**, ni vos **missions**, ni les **diapositifs** du cours, ni les **capsules vidéo**, ni vos **notes**
- Vous ne pouvez utiliser **aucun autre document ou logiciel** dont vous disposez ou disponible sur le web
- Vous vous engagez à faire votre examen de manière strictement individuelle, **sans assistance extérieure ni communication** avec d'autres étudiants.
- Un système de **détection de plagiat** sera appliqué à cet examen.
- Vous devrez accepter ces consignes de confidentialité avant l'examen (sur le site INGInious de l'examen).

# [Q1] Manipulation de listes

Collapse context

Définissez une fonction `mix(l)` qui prend comme entrée une liste avec un nombre **pair** d'entiers, par exemple : [1,2,3,4, ..., ,97,98,99,100]. La fonction retourne une nouvelle liste de la même taille, par exemple : [1,100,2,99,3,98, ...,50,51], avec comme premier élément le premier élément de la liste originale, comme deuxième élément le dernier élément de la liste originale, comme troisième élément le deuxième élément de la liste originale, comme quatrième élément l'avant-dernier élément de la liste originale...

Plus précisément, pour chaque index *i* pair où $0 \leq i < n$ et *n* est la taille de la liste originale, les éléments aux positions *i* et *i+1* de la nouvelle liste sont...

La fonction demandée est la suivante:

```
def mix(l):
    """
    @pre:    l est une liste d'entiers
             la taille n de cette liste est un nombre pair
    @post:   retourne une liste r d'entiers
             la liste retournée r a la même taille n
             pour chaque index 0 ≤ i < n où i est pair on a la
             correspondance suivante entre les deux listes :
                 r[i] = l[i//2]
                 r[i+1] = l[n-1-(i//2)]
    """
```

Si nécessaire vous pouvez aussi définir des fonctions auxiliaires.

## Question 1: mix

```
1  def mix(l):    # Ne pas effacer cette ligne
2      """
3      @pre:    l est une liste d'entiers
4               la taille n de cette liste est un nombre pair
5      @post:   retourne une liste r d'entiers
6               la liste retournée r a la même taille n
7               pour chaque index 0 ≤ i < n où i est pair on a la
8               correspondance suivante entre les deux listes :
9                   r[i] = l[i//2]
10                  r[i+1] = l[n-1-(i//2)]
11     """
```

## Question 2: Zone de test

Insérez votre code de test optionnel ci-dessous. Vous pouvez utiliser `print` pour tester votre programme. Votre code de test sera exécuté à la suite de votre méthode `mix` et d'éventuelles fonctions auxiliaires ci-dessus (il ne faut pas les recopier ici).

Exemple:

```
1  print(mix([1, 2, 3, 4]))
2  # [1, 4, 2, 3]
3
4  # vos tests
```

Submit

Contexte d'une question...

Détails de la question...

Réponse à la question...

Zone test Python pour exécuter du code de test, tenant compte du code de votre solution...

Soumettre votre réponse et lancer votre code de test...

# Evaluation de l'examen

- **Durant** l'examen
  - soumettre exécute uniquement ***vos*** tests, pas les notres
  - 100% vert = le code est exécutable ≠ score de 100%

    Your answer passed the tests! Your score is 100.0%.
  - 0% rouge = votre code ne fonctionne pas : erreurs de Python

    There are some errors in your answer. Your score is 0.0%. => score de 0 %
- **Après** l'examen : évaluation avec ***nos*** tests
  - correction automatisée avec INGInious
  - pas de correction manuelle
  - une solution exécutable et partiellement correcte peut obtenir des points
  - une solution qui n'exécute pas donnera un score de 0%
  - Outil de détection de plagiat

*Est ce que notre code sera encore corrigé manuellement?*

# Evaluation de l'examen

Quelles sont les bonnes pratiques standards à appliquer dans notre création de code?

Lors de l'examen, si notre code ne passe pas tous vos tests, regardez-vous le code afin de savoir où est le problème ?

Doit-on mettre des commentaires, des spécifications pre-post, etc. ?

- La correction est ***complètement automatisé***
  (nos tests d'évaluation)
  - Vérifie le input-output de votre programme
  - Vérifie les cas de bords
  - Peut vérifier la structure interne de votre programme
  - Et même l'efficacité d'un programme
  - Ne lit pas vos commentaires
    et donc pas vos spécifications pre-post
  - Vérifie surtout le bon comportement de votre solution

# Evaluation de l'examen

Faut-il utiliser la programmation défensive pour vérifier les pré- et postconditions?

- Le principe d'une précondition est que vous pouvez assumer qu'elle sera satisfaite quand on appelle la fonction/méthode
- Donc pas besoin de la vérifier explicitement dans votre fonction/méthode
- Par contre, bien vérifier dans vos *tests* que la postcondition est satisfaite

Est-ce qu'on pourra utiliser des méthodes Python qui n'ont pas été présentées explicitement au cours ?

- Chaque solution est réalisable avec ce que vous avez vu au cours
- Mais tout le langage Python reste évidemment à votre disposition
- Néanmoins, *dans aucune question il ne sera possible de faire des import.*

# La note du cours

```
def score(interro, examen, bonus):  # /20, /20, /1
    if examen > interro :
        note = examen + bonus
    else:
        note = (1/3 * interro) + (2/3 * examen) + bonus
    if note > 20 :
        note = 20
    return round(note)

>>> score(15, 6, 0)
9
>>> score(10, 15, 1)
16
>>> score(20, 20, 1)
20
```

*Pourriez-vous expliquez comment la note finale sera calculée?*

# Matière à connaître

- Il n'y aura pas de questions QCM sur la théorie
- Possibilité de faire des tests en Python
  - code de test en Python
  - fonction help() de Python
- Il ne faut ***pas*** connaître par cœur toutes les fonctions de Python
  - Le syllabus théorie sera disponible
    (lien via INGInious)
- Quelle matière à (ne pas) connaître?

# Matière – Partie 1

| À connaître | Pas à connaître |
|---|---|
| **Syntaxe** de Python | |
| **Expressions** et **instructions** | |
| **Opérateurs** arithmétiques et **ordre** | |
| **Variables** et **affectation** | |
| Boucles : **for**, **while**, **range**() | Turtle graphics |
| Types de données : **int**, **float**, **bool**, **str** | |
| **Conditionnelles** et valeurs **booléennes** | |
| **Fonctions** : définition, paramètres, arguments, résultats | |
| **Portée** et **durée de vie** des variables | |
| Spécifications pré/post | Définition de modules |
| Modules et import | Modules particuliers : random, time, os, … |

# Matière – Partie 2

| À connaître | | Pas à connaître |
|---|---|---|
| **Strings** | notation [i], [:i], [i:j], [i:] | format() |
| | len(), +, * | |
| | strip(), split(), upper(), lower() | find(), join () |
| | >, <, >=, <=, ==, != | |
| | for … in … | if … in, not in |
| **Listes** | notation [i], [:i], [i:j], [i:] | listes en compréhension |
| | len(), +, * | enumerate() |
| | append() | sort(), sorted(), extend() |
| | remove() ≠ del | insert(), count(), index(), reverse() |
| | for … in … | if … in, not in |
| string → liste | list() | |
| … → string | str() | |
| **Tests** | assert | if__name__== '__main__' |

# Matière – Partie 2

| À connaître | | Pas à connaître |
|---|---|---|
| **Tuples** | notation () et [i] | |
| **Dictionnaires** | notation { : } et [ ] | keys(), values() |
| | notation [], get(), in | setdefault() |
| | for … in, items() | |
| Références | | copy ≠ deepcopy |
| Structures de données imbriquées et hétérogènes | | |
| Structures muables vs. immuables | | |
| **Fichiers** | open, close, with ... as | |
| | read, readline, readlines | |
| | write, writelines | |
| | for ... in file | |
| **Exceptions** | try... except, try ... except as | les différents types d'exceptions : IndexError, FileNotFoundError, … |
| | raise | |

# Matière – Partie 3

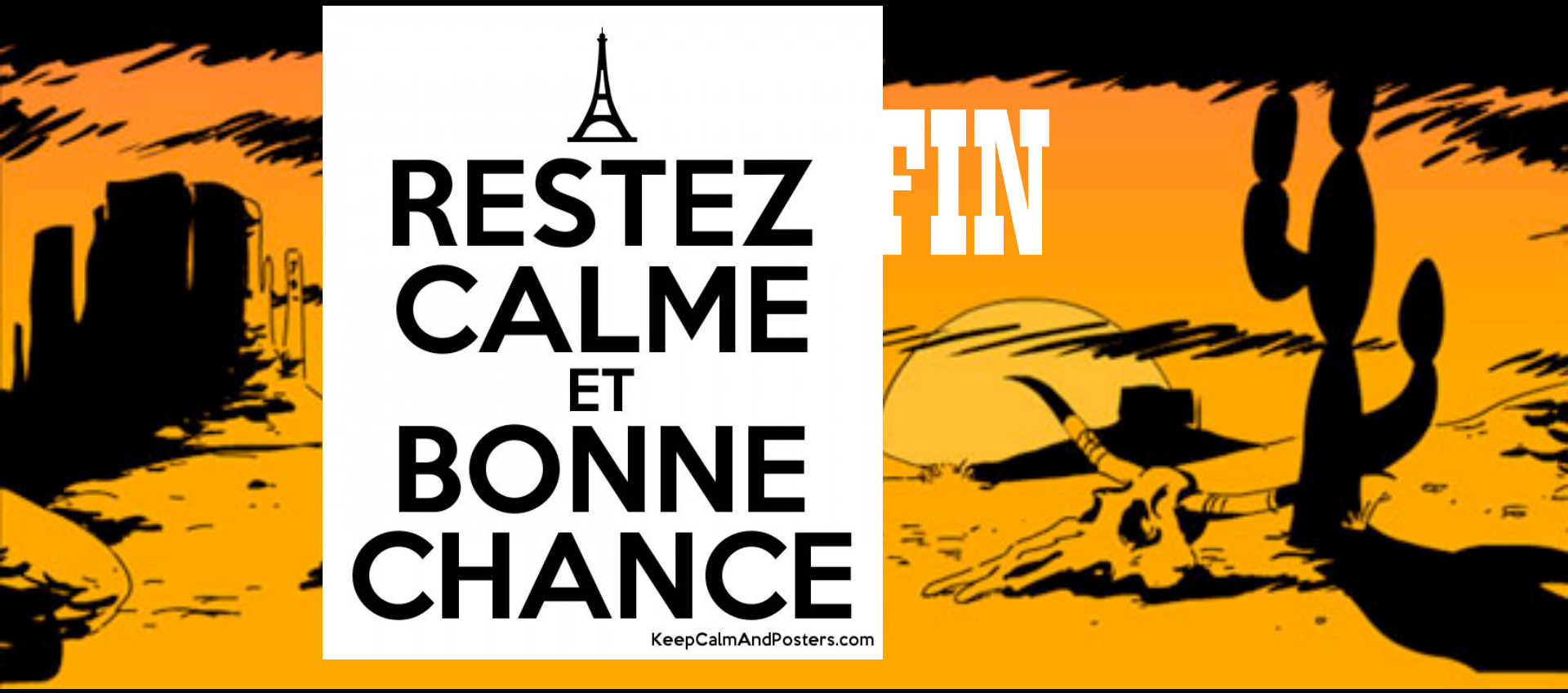
| À connaître | Pas à connaître |
|---|---|
| is versus == | |
| classes et objets | |
| attributs et méthodes d'instance | |
| création d'un objet | |
| appel d'une méthode | |
| appel à self ( self.attribut, self.method() ) | |
| références | copy, deepcopy |
| égalité entre objets | |
| __init__, __str__, __eq__ | les autres méthodes magiques |

# Matière – Partie 3

| À connaître | Pas à connaître |
|---|---|
| attributs privés | méthodes privées |
| méthodes accesseurs et mutateurs | |
| attributs et méthodes de classe | |
| composition de classes | classes imbriquées |
| héritage | héritage multiple |
| super() | |
| polymorphisme | tests unitaires avec unittest |
| la liaison dynamique de self | |
| portée et visibilité des variables | |

# Matière – Structures et algorithmes

| À connaître | Pas à connaître |
| --- | --- |
| Principes de comment un algorithme de recherche dichotomique fonctionne | Connaître par cœur le code d'un algorithme de recherche dichotomique |
| Comment utiliser une structure chainée | Connaître par cœur l'implémentation de la classe LinkedList |
| Comment implémenter une liste chaînée | |


FIN
RESTEZ
CALME
ET
BONNE
CHANCE
KeepCalmAndPosters.com