# Notions de base

Vendredi 19 septembre

Option Informatique

Ecole Alsacienne

# AVANT DE COMMENCER

- Avez-vous vos identifiants pour vous connecter ?

- Des nouveaux venus ?
  - Slides du cours d'introduction : http://andre.lovichi.free.fr/teaching/
  - Fiche de renseignements à remplir

- D'autres questions ?

# Programme de la séance

1. Langages et algorithmes
2. Python et pseudo-code
3. Variables et typage
4. Constructions classiques
5. Fonctions et arguments
6. Exercices

# Algorithmes et Programmes

# UNE RECETTE DE CUISINE

Leçon du jour : le crumble aux pommes

# Une recette de cuisine

1. Eplucher les 6 pommes, en retirer les pépins, et les couper en petits morceaux

2. Faire cuire les morceaux de pommes à la poêle avec 50 g de sucre et un peu de cannelle

3. Faire préchauffer votre four à 180 °C et beurrer un plat

4. Mélanger dans un saladier 150g de farine, 100g de sucre et 75g de beurre pour obtenir un mélange sableux

5. Mettre les pommes dans le plat et verser le mélange par-dessus, et laisser cuire au four pendant 30 minutes.

# Une recette de cuisine

1. Eplucher les 6 pommes, en retirer les pépins, et les couper en petits morceaux

2. Faire cuire les morceaux de pommes à la poêle avec 50 g de sucre et un peu de cannelle

3. Faire préchauffer votre four à 180 °C et beurrer un plat

4. Mélanger dans un saladier 150g de farine, 100g de sucre et 75g de beurre pour obtenir un mélange sableux

5. Mettre les pommes dans le plat et verser le mélange par-dessus, et laisser cuire au four pendant 30 minutes.

# ALGORITHME

- Une description précise d'une méthode de résolution
- Une suite de tâches élémentaires connues de l'utilisateur qui va devoir les effectuer
- Caractéristiques requises :
  - Non-ambiguïté : il n'y a aucune initiative à prendre
  - Exhaustivité : tous les cas de figure sont prévus
  - Terminaison : on a toujours un résultat

# La communication humaine

*Au fur et à mesure que vous lisez cette phrase (un peu longue), vous identifiez inconsciemment les différents éléments qui la composent, ce qui vous permettre de donner du sens à cette suite de symboles.*

- La langue française est un moyen de communication, avec ses règles, ses codes, et son alphabet

- Lors que vous y êtes confronté, par écrit ou par oral, vous en déduisez "naturellement" le sens

# Langages

- Il existe différents types de langages, dont tout le monde n'a pas la même maitrise :
  - Do you speak english ?
  - 你说汉语吗 ?
  - 
- Même s'il y a des erreurs dans une phrase, on parvient à en retrouver le sens
  - Exemple : *Sleon une edtue de l'Uvinertise de Cmabrigde, l'odrre des ltteers dnas les mtos n'a pas d'ipmrotncae*

# Langages et machines

- Le problème est que les ordinateurs d'aujourd'hui sont beaucoup moins doués qu'un humain
  - Ils ne maitrisent ni le français, ni l'anglais, ni le chinois, ni l'elfique
  - S'il y a une erreur quelque part, ils sont perdus

- Pour contrôler un ordinateur, il est donc important de
  - Lui parler avec des mots simples
  - Dans une syntaxe clairement définie
  - Sans faire d'erreur

# Algorithme et programme

- Un programme, c'est la traduction d'un algorithme dans un langage compréhensible par une machine

- Il existe donc plusieurs programmes possibles pour un même algorithme
  - En fonction du langage de programmation choisi
  - En fonction des outils utilisés dans le programme

# Python et pseudo-code

# Ecrire un algorithme

- Solution universelle : le pseudo-code

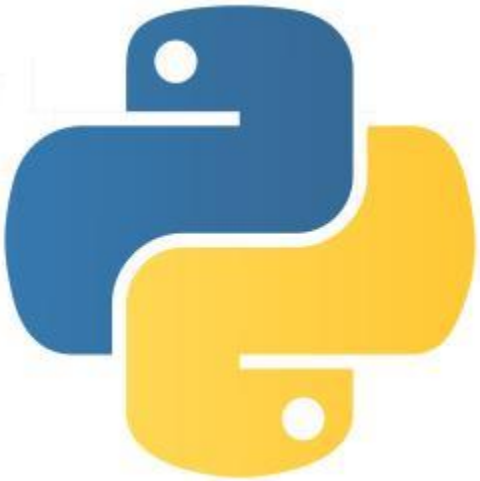
- Exemple :

```
Pour chaque pomme,
      Faire
            Eplucher(pomme)
            RetirerPépins(pomme)
            CouperEnMorceaux(pomme)
      Fin faire
```

- Avantage : on peut ensuite adapter (assez) facilement ce pseudo-code à un langage de programmation précis

# Un langage précis

Cette année, nous travaillerons en Python



Depuis 2013, c'est le langage de référence pour la matière « informatique pour tous » en classe préparatoire

# WHAT IS YOUR FAVORITE LANGUAGE ?

- Langage créé à la fin des années 90 par Guido van Rossum

- Nom tiré des Monty Python

# Avantages de Python

- Conçu pour être lisible
- Syntaxe simple (tout en imposant un minimum de rigueur)
- Gestion automatique de la mémoire (garbage collector)
- Quelques entreprises qui utilisent Python :
  - Google
  - NASA
  - Blender
  - etc.

# Et Caml ?

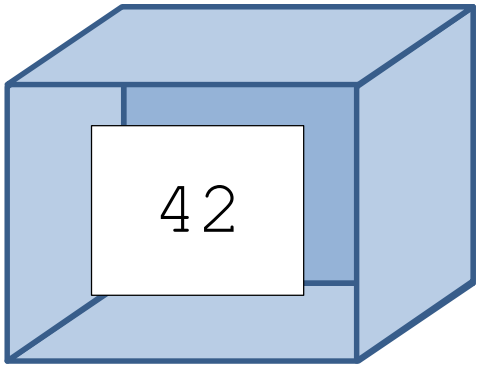
- Vous avez peut-être aussi entendu parler de Caml ou Ocaml


- Ce langage est également utilisé en classe préparatoire, mais plutôt pour l'option informatique.

# Variables et Typage

# Qu'est-ce qu'une variable ?

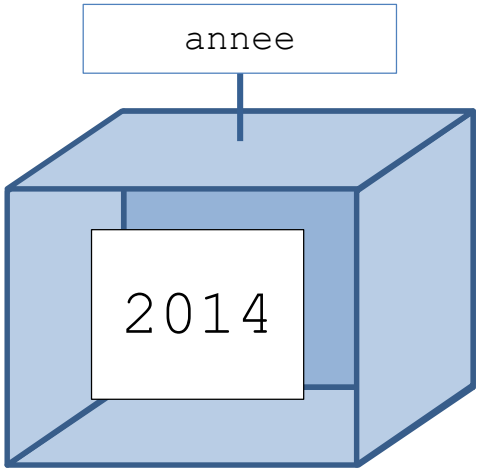
- Une variable est un objet stocké par l'ordinateur
- Informellement, c'est une sorte de boîte




- Et on va pouvoir stocker un élément à l'intérieur

# Le nom d'une variable

- Une variable est désignée par son nom :
  - Choisir ce nom intelligemment
  - Eviter les espaces, les accents et les caractères spéciaux
  - Utiliser éventuellement des "traits bas" (underscore) (`par_ex`)
  - Jamais deux variables avec le même nom au même moment !



# Déclaration de variable

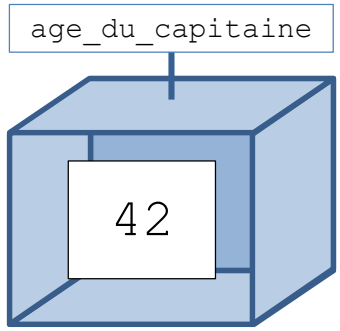
- Une **déclaration** de variable consiste à indiquer à l'ordinateur la création d'une variable pour qu'il prenne les mesures nécessaires (notamment l'allocation de mémoire)

- Pseudo-code :

  `age_du_capitaine ← 42` (nom pas encore utilisé)

- Python :

```
age_du_capitaine = 42
```



# Accès à une variable

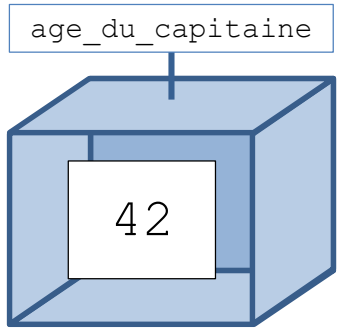
- Pour accéder au contenu d'une variable, il suffit d'utiliser son nom, et l'ordinateur se chargera de récupérer la valeur correspondante.

- Pseudo-code :

  `age_du_capitaine` (correspond au nombre 42)

- Python :

  `age_du_capitaine` (correspond au nombre 42)



# CHANGER LA VALEUR D'UNE VARIABLE

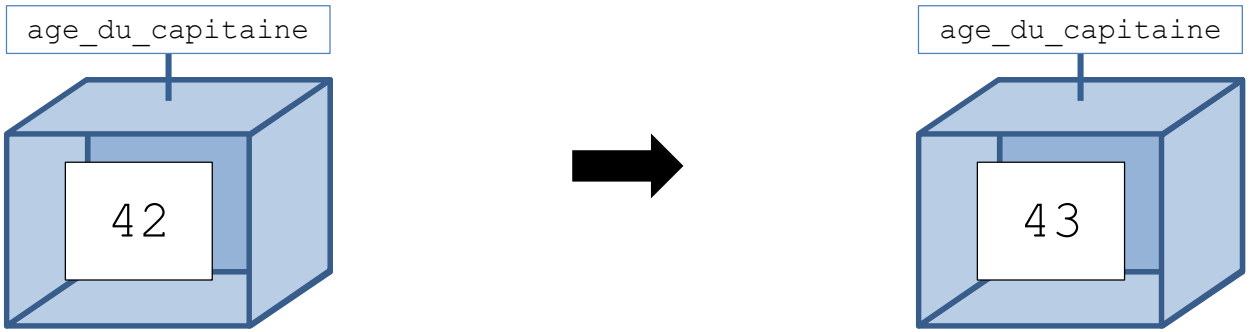
- L'affectation consiste à donner une nouvelle valeur à une variable

- Pseudo-code :

  `age_du_capitaine` ← `43` (nom déjà existant)

- Python :

```
age_du_capitaine = 43
```



# Petit exercice

- Voici quelques lignes de pseudo-code

```
x ← 10
y ← x + 5
x ← 1
z ← y - 3
```

- **Question 1 :** Transcrivez ces lignes en Python

```
x = 10
y = x + 5
x = 1
z = y - 3
```

- **Question 2 :** Que vaut `z` ?

# Qu'est-ce que le typage ?

≠

# Qu'est-ce que le typage ?

$$42 \neq \texttt{"bonjour"}$$

# Qu'est-ce que le typage ?

42 ≠ "quarante deux"

# Qu'est-ce que le typage ?

$$42 \neq 42,0$$

# Les types les plus classiques

- Dès sa déclaration, chaque variable se voit affecter un type précis, qui caractérise le type d'information qu'elle contient

- Les types les plus classiques sont :
  - `int` : entier relatif (positif ou négatif) (ex : `-13`)
  - `float` : nombre réel (avec virgule) (ex : `3.14`)
  - `str` : chaîne de caractères (ex : `"bonjour"`)
  - `bool` : booléen (ex : `true` ou `a>3`)
  - `NoneType` : rien (équivalent de `void` ou `unit`)

- Remarque : Dans certains langages (mais pas en Python), il existe aussi un type `char`, qui correspond à un caractère.

# Un petit test

*Quel est le type de chacun des éléments ci-dessous ?*

```
4.0
```

```
4>0
```

```
4
```

```
"4"
```

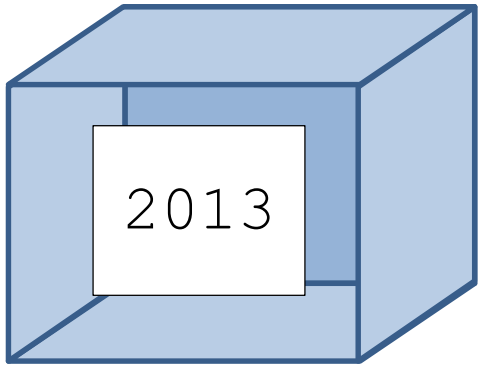
# Types complexes

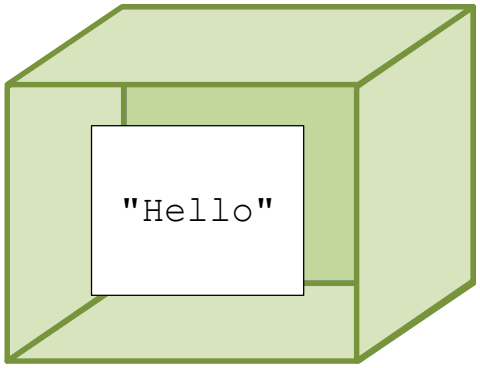
Python connait de nombreux autres types :

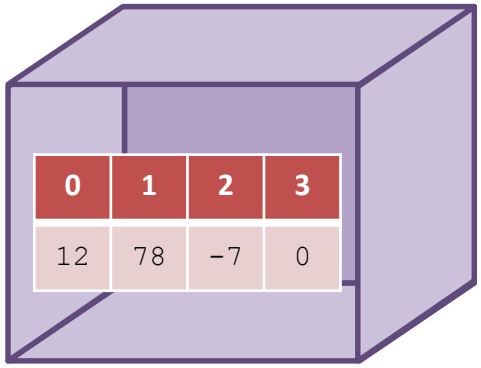
- Nombres complexes : `(4+3j)` de type `complex`
- *n*-uplets : `(1, "ab", true)` de type `int * str * bool`
- Tableaux : `[ 3 , 6 ]` de type `list`
- Structures fournies dans les autres modules de Python
- Tout ce qu'on aura envie de créer

# Différents types de boîtes

- Lorsqu'une variable est créée, un type précis lui est affecté




- Si l'on essaye de manipuler cette variable en se trompant sur le type, Python nous le fera savoir :

```
x = "Hello"
y = x + 3

TypeError: Can't convert 'int' object to str implicitly
```

# CONSTRUCTIONS CLASSIQUES

# Séquence

- Une **séquence** d'instructions permet d'exécuter plusieurs instructions l'une après l'autre

- Pseudo-code :

```
instruction1
instruction2
```

- En Python, il suffit de passer à la ligne

```
instruction1
instruction2
```

- ⚠ En Python, la mise en forme est très importante ! ⚠

# Séquence

- Dans une séquence, chaque instruction n'est exécutée qu'une fois que la précédente a été traitée

- On peut donc utiliser les résultats d'une instructions dans la suivante :

```
x = 24
resultat1 = gros_calcul_complexe(x)
resultat2 = autre_calcul_complexe(resultat1)
```

- La plupart des langages permettent de faire des calculs en parallèle : plusieurs instructions sont alors traitées en même temps (par exemple pour améliorer les performances).

# Itération

- Parfois, il est nécessaire d'effectuer plusieurs fois la même action, voire la même suite d'actions : on parle d'itération, ou de boucle.

```
Pour chaque pomme,
     Faire
          Eplucher(pomme)
          RetirerPépins(pomme)
          CouperEnMorceaux(pomme)
     Fin faire
```

# Boucles

Dans la plupart des langages, il existe deux types de boucles :

- Boucle `for`
  - Exécuter *n* fois un bout de code, ce nombre *n* étant défini à l'avance
  - Bien adapté aux tableaux et aux chaînes de caractères
- Boucle `while`
  - Exécuter un bout de code tant qu'une certaine condition n'est pas remplie (un test est effectué à chaque fois)
  - Bien adapté aux structures récursives
  - Risque de boucle infinie (le programme ne s'arrête jamais)

# Boucle `for`

- Une boucle `for` commence par la déclaration d'un indice et de deux bornes (de type `int`) entre lesquelles cet indice va progressivement évoluer.

- Sauf indication contraire, après chaque passage dans le corps de boucle, l'indice est incrémenté (on lui ajoute 1).

- Lorsque l'indice atteint la deuxième borne, la boucle s'arrête, et on passe à la suite du programme
  - Attention, en général, la borne supérieure n'est donc pas incluse

# UNE BOUCLE FOR EN PSEUDO-CODE

- Formulation propre en pseudo-code :

```
Pour p allant de 1 (inclus) à 7 (exclus)
    Faire
            Eplucher(p)
            RetirerPépins(p)
            CouperEnMorceaux(p)
    Fin faire
```

- Le corps de boucle se situe entre `Faire` et `Fin faire`

# UNE BOUCLE `for` EN PYTHON

- La même chose en Python :

```
for p in range(1,7):
    eplucher(p)
    retirer_pepins(p)
    couper_en_morceaux(p)
```

- Le corps de boucle est indiqué par l'indentation (décalage de trois espaces sur la droite)
- ⚠ En Python, la mise en forme est très importante ! ⚠
- Une boucle `for` est de type `NoneType` (on ne renvoie rien)

# VARIANTES CLASSIQUES

- Si la borne de départ est 0, il n'est pas obligatoire de la renseigner :

```
for p in range(6):
    eplucher(p)
    retirer_pepins(p)
    couper_en_morceaux(p)
```

Valeurs successives de p : $\{0, 1, 2, 3, 4, 5\}$

- Il est également possible de choisir le pas, c'est-à-dire le décalage effectué sur l'indice à chaque tour de boucle :

```
for p in range(5,35,5):
    eplucher(p)
    retirer_pepins(p)
    couper_en_morceaux(p)
```

Valeurs successives de p : $\{5, 10, 15, 20, 25, 30\}$

- On peut également faire décroitre l'indice (avec un pas négatif)

```
for p in range(10,0,-1):
    eplucher(p)
    retirer_pepins(p)
    couper_en_morceaux(p)
```

Valeurs successives de p : $\{10, 9, 8, 7, 6, 5, 4, 3, 2, 1\}$

# Boucle `while`

- Une boucle `while` dépend avant tout d'une condition, c'est-à-dire une question fermée (à laquelle on répond par oui ou par non).

- Au début de chaque passage dans la boucle, on vérifie que cette condition est vérifiée :
  - Si c'est le cas, on exécute le corps de boucle et on revient tester la condition
  - Sinon on passe à la suite du programme

# UNE BOUCLE **WHILE** EN PSEUDO-CODE

- Formalisation propre en pseudo-code :

```
Tant que (il reste une pomme à couper)
    Faire
        p ← ChoisirUnePomme()
        Eplucher(p)
        RetirerPépins(p)
        CouperEnMorceaux(p)
    Fin faire
```

- Le corps de boucle se situe entre `Faire` et `Fin faire`

# UNE BOUCLE `while` EN PYTHON

- La même chose en Python :

```
while (nombre_pommes != 0):
    p = choisir_une_pomme()
    eplucher(p)
    retirer_pepins(p)
    couper_en_morceaux(p)
    nombre_pommes = nombre_pommes - 1
```

- Le corps de boucle est indiqué par l'indentation (décalage de trois espaces sur la droite)

- ⚠ En Python, la mise en forme est très importante ! ⚠

- Une boucle `while` est de type `NoneType` (on ne renvoie rien)

# QUAND UTILISER UNE BOUCLE `WHILE` ?

- Cette construction se combine bien avec des structures définies récursivement

- La condition est en fait un booléen :
  - soit un test booléen (ex : `prenom == "Bob"` ou `age < 18`)
  - soit une fonction qui renvoie un booléen

# ATTENTION AUX BOUCLES INFINIES

- Si le test d'une boucle `while` renvoie toujours vrai, le programme reste indéfiniment dans cette boucle

- Exemple idiot :

```
x = 0
while (1 < 2):
    x = x + 1
```

- Exemple un peu moins idiot :

```
x = 0
while (x < 10):
    x = x + 1
```

```
x = 0
while (x < 10):
    x = x - 1
```

- Conclusion : attention avec les boucles `while` !

# Prétraitement et posttraitement

- Il est fréquent de rencontrer des séquences d'instructions composées de :
  - Un pré-traitement
  - Une boucle (`for` ou `while`)
  - Un post-traitement

- Exemple :
  - Comment calculer la température moyenne de la semaine précédente ?
  - On suppose qu'on dispose d'une fonction `temperature` telle que `temperature(i)` est la température du $i^{ème}$ jour
  - Solution :

```
somme ← 0
Pour jour allant de 0 (inclus) à 7 (exclus)
   Faire
      somme = somme + temperature(jour)
   Fin faire
moyenne ← somme / 7
```

# Exercices

- **Question 1** : Réécrivez ce programme en utilisant une boucle `while` :

```
Pour indice allant de 0 (inclus) à 10 (exclus)
    Faire
        Afficher(indice)
    Fin faire
```

- **Réponse** :

```
indice ← 0
Tant que (indice < 10)
    Faire
        Afficher(indice)
        indice ← indice + 1
    Fin faire
```

- **Question 2** : Transposez ces deux programmes en Python

# Les conditions

- Parfois, plusieurs cas de figures sont possibles, et l'action à entreprendre dépend de la situation :

```
Si (Le feu est rouge)
    Alors
        S'arrêter
    Sinon
        Passer
```

# ECRITURE EN PSEUDO-CODE

- Pseudo-code :

```
Si (condition)
    Alors
            Instruction1
    Sinon
            Instruction2
Fin si
```

# En Python : `if ... then ... else`

- En Python :

```
if (condition):
    instruction1
else:
    instruction2
```

- La `condition` est de type `bool`

- Les instructions à exécuter dans les deux cas sont indiquées par l'indentation (décalage de trois espaces sur la droite)

- ⚠ En Python, la mise en forme est très importante ! ⚠

# SANS LE **ELSE**

- En pseudo-code :

```
Si (condition)
    Alors
            instruction1
Fin si
```

- En Python :

```
if (condition):
    instruction1
```

# Un problème ?

- Que fait le programme suivant ?

```
if (condition):
    instruction1
else:
    instruction2
instruction3
```

- Et celui-ci ?

```
if (condition):
    instruction1
else:
    instruction2
    instruction3
```

# IMBRICATIONS

- Il est possible d'utiliser plusieurs conditions les unes dans les autres :

```
if (conditionA):
    instruction1
else:
    if (conditionB):
        instruction2
    else:
        instruction3
```

- ⚠ En Python, la mise en forme est très importante ! ⚠

# Exercice

- **Question 1** : Que fait ce programme ?

```
x = 25
a = 0
recherche = true

while (recherche):
    if (a*a == x):
        recherche = false
    else:
        a = a + 1

print(a)
```

- **Question 2** : Quel est le problème si on change la valeur de x ?

- **Question 3** : Comment pourrait-on le résoudre ?

# Fonctions et arguments

# FONCTIONS

- Une fonction est une suite d'actions définie dans un programme afin de pouvoir y faire appel quand on veut

- Elle est caractérisée par :
  - Son nom
  - Ses arguments : le nombre et le type d'objet qu'elle prend en entrée
  - Son contenu : la suite d'actions qu'elle effectue à partir de ces objets
  - Sa sortie : le résultat qu'elle renvoie quand elle a fini

# Un exemple en pseudo-code

- Fonction de calcul

```
CalculerCarreNombre(n) =
    carre ← n*n
    Renvoyer carre
```

- Fonction principale

```
Main() =
    Pour i allant de 1 (inclus) à 11 (exclus)
       Faire
          c = CalculerCarreNombre(i)
          Afficher(c)
       Fin faire
    NeRienRenvoyer()
```

# Le même exemple en Python

- Fonction de calcul

```
def calcul_carre_nombre(n):
    carre = n*n
    return carre
```

- Fonction principale

```
def main() =
    for i in range(1,11):
        c = calcul_carre_nombre(i)
        print(c)
    return
```

- ⚠ En Python, la mise en forme est très importante ! ⚠

# RÉCURSIVITÉ : L'EXEMPLE DE LA FACTORIELLE

- Définition

$$n! = factorielle(n) = \begin{cases} 1 \; si \; n = 0 \\ 1 \times 2 \times \cdots \times n \; sinon \end{cases}$$

- Remarque :

$$\text{Pour n} > 0, factorielle(n) = n \times factorielle\,(n-1)$$

- Pseudo-code :

```
Factorielle(n) =
    Si (n = 0)
       Alors
          Renvoyer 1
       Sinon
          Renvoyer n * Factorielle(n-1)
    Fin si
```

# "L'ÉTERNITÉ, C'EST LONG, SURTOUT VERS LA FIN" (W. ALLEN)

Ce qu'il ne faut pas faire

**Faire un crumble** =

1. Eplucher les 6 pommes, en retirer les pépins, et les couper en petits morceaux
2. Faire cuire les morceaux de pommes à la poêle avec 50 g de sucre et un peu de cannelle
3. **Faire un crumble**
4. Faire préchauffer votre four à 180 °C et beurrer un plat
5. Mélanger dans un saladier 150g de farine, 100g de sucre et 75g de beurre pour obtenir un mélange sableux
6. Mettre les pommes dans le plat et verser le mélange par-dessus, et laisser cuire au four pendant 30 minutes.

# Fonctions récursives

- Une fonction **récursive** est une fonction qui s'appelle elle-même

- Pour éviter de boucler à l'infini, l'appel récursif doit se faire sur un argument différent de celui de la fonction appelante

- Une fonction récursive doit comporter :
  - Un (ou plusieurs) cas de base
  - Un (ou plusieurs) cas récursif(s)

# Les fonctions récursives en Python

- En Python, une fonction peut faire appel à elle-même : elle devient alors récursive.

- Exemple :

```
def factorielle(n):
    if (n == 0):
        return 1
    else:
        return n * (factorielle (n-1))
```

# Qu'est-ce qu'un argument ?

Trois définitions plus ou moins rigoureuses :

- Le truc qu'on "donne à manger" à la fonction
- Un des éléments sur lequel on applique la fonction
- Une variable locale à la fonction dont la valeur n'est pas connue à l'avance

# Exemples

- Calcul du carré d'un nombre :

```
CalculerCarreNombre(n) =
    carre ← n*n
    Renvoyer carre
```

- Fonction de Syracuse :

```
Syracuse(n) =
    Si (n est pair)
        Alors
            Renvoyer n/2
        Sinon
            Renvoyer 3*n+1
    Fin si
```

$$Syracuse(n) = \begin{cases} n/2 & si\ n\ est\ pair \\ 3 \times n + 1 & si\ n\ est\ impair \end{cases}$$

# Exercices

# Premières fonctions

- Pour chacune des fonctions suivantes, écrivez une version récursive et une version non récursive

1. Une fonction `somme1` telle que `somme1 n` renvoie la somme des entiers positifs inférieurs ou égal à n (1 + 2 + … + n)

2. Une fonction `somme2` telle que `somme2 a b` renvoie la somme des entiers compris entre a et b (a + (a+1) + (a+2) + … + (b-1) + b)

3. Une fonction `produit` telle que `produit n` renvoie le produit des entiers strictement positifs et inférieurs ou égaux à n (1 * 2 * … * n )

4. Une fonction `factorielle` telle que `factorielle n` renvoie la factorielle de n

# QUELQUES PETITS EXERCICES POUR FINIR…

- Le coefficient binomial $\binom{n}{p}$ correspond au nombre de choix possibles de $p$ éléments parmi $n$ (sans notion d'ordre)

- Propriétés :
  - $\binom{n}{p} = \frac{n}{1} \times \frac{n-1}{2} \times \cdots \times \frac{n-p+1}{p} = \frac{n!}{(n-p)! \times p!}$
  - $\binom{n}{p} + \binom{n}{p+1} = \binom{n+1}{p+1}$

- Exercice:
  1. Imaginez une fonction non récursive pour calculer le coefficient $\binom{n}{p}$
  2. Imaginez une fonction récursive pour calculer le coefficient $\binom{n}{p}$
  3. Essayez d'estimer le nombre d'appels récursifs de cette deuxième fonction. Comment pourrait-on le réduire ?

# PROCHAINE SÉANCE

Vendredi 26 septembre

## [TD] BASES DE PROGRAMMATION