Feuille de cours 5 (informatique) : Listes

Une liste est une séquence d'éléments de types quelconques, rangés dans un certain ordre, listés entre crochets et séparés par des virgules. Voici quatre exemples de listes :

```
L1 = [1,2,3]
L2 = [3,2,1]
L3 = [1,1,1,1,1]
L4 = [3,'banane',[1,2]]
```

Remarque 1

- l'ordre des éléments compte (ainsi pour Python `L1` $\neq$ `L2`)
- un élément peut être présent plusieurs fois dans une liste (comme dans `L3`)
- les éléments d'une liste ne sont pas forcément de même type (comme dans `L4`)

La strucure de liste est pratique pour stocker un grand nombre de données sans avoir besoin de créer une variable pour chacune d'entre elles.

Important : on sera souvent amené à utiliser la **liste vide** c'est-à-dire la liste ne contenant aucun élément : il s'agit de `L = []`.

1 Création de listes

Lorsqu'une liste ne contient que peu d'éléments, tous connus à l'avance, on peut la saisir directement "à la main", par exemple pour créer une liste `L` correspondant au 5-uplet $(3, -2, 7, 0, 1)$ on écrira :

En pratique cependant, on est le plus souvent amené à demander à Python de créer lui-même la liste. Pour cela, il existe essentiellement deux méthodes.

1.1 Création par ajouts successifs d'éléments

Pour créer une liste `L`, on peut :

- initialiser `L` à la liste vide `[]`, puis
- lui rajouter successivement des éléments grâce à une boucle `for`.

Important : pour rajouter un élément `x` à une liste `L`, on utilise la commande `L.append(x)`. Avec cette commande, l'élément `x` est rajouté **à la fin** de la liste `L`.

```
L = []
for ... :
  L.append(element a ajouter)
```

Remarque 2

Notez que la commande `L.append(x)` ne crée pas une nouvelle liste, mais *modifie* la liste `L` (ainsi, il ne faut **pas** écrire `L = L.append(x)` mais simplement `L.append(x)`).

Exercice 1

Que contient `L` après exécution du code ci-contre ?

```
L = []
L.append(4)
L.append(7)
L.append(2)
```

Que contient `M` après exécution du code ci-contre ?

```
M = []
for k in range(4):
    M.append(k)
```

Exercice 2

1. Écrire une fonction prenant en argument un entier `n` supérieur à 3 et renvoyant la liste `L = [3,4,5,...,n]`.
2. Écrire une fonction prenant en argument `n` $\in \mathbb{N}$ et renvoyant la liste `L = [1,4,9,16,...,`n^2`]`.

1.2 Création "en compréhension"

Il existe une deuxième façon de créer des listes en donnant la forme de leurs éléments, c'est d'utiliser la syntaxe

`[f(k) for k in L]`

où `L` est elle-même une liste ou un range, et où `f(k)` désigne n'importe quelle expression dépendant (ou non) de `k`.

Par exemple, `[k**2+2 for k in range(4)]` vaut

Exercice 3

Comment créer la liste `L2 = [1,4,9,...,`n^2`]` à partir de la liste `L1 = [1,2,3,...,n]` ?

Exercice 4
Définir une liste `Lbis` contenant les mêmes éléments que `L` mais créée par une autre méthode :

```
L=[]
for k in range(10):
    L.append(2*k)
```

```
L=[k**2 for k in range(2,7)]
```

1.3 Liste des termes d'une suite

Il arrive très souvent qu'on veuille obtenir la liste des termes succcessif d'une suite c'est-à-dire une liste du type `L = [`u_0`,`u_1`,`u_2`, ...,`u_n`]`.

Quelle méthode utiliser si la suite (u_n) est donnée de manière explicite c'est-à-dire sous la forme $(u_n) = (f(n))$?

Quelle méthode utiliser si la suite (u_n) est définie par récurrence par une relation du type $u_{n+1} = f(u_n)$ ou $u_{n+1} = f(u_n, n)$?

Exercice 5
Dans chacun des cas suivants, écrire une fonction prenant en argument n et renvoyant la liste `[`u_0`,`u_1`,`u_2`, ...,`u_n`]`.

1. $\forall n \in \mathbb{N},\ u_n = \sqrt{n} + 1$
2. $u_0 = 2$ et $\forall n \in \mathbb{N},\ u_{n+1} = \sqrt{u_n} + 1$
3. $u_0 = 3$ et $\forall n \in \mathbb{N},\ u_{n+1} = \dfrac{n}{1 + u_n}$.

2 Premières opérations sur les listes

Concaténation :
On peut concaténer deux listes `L1` et `L2`, c'est-à-dire obtenir la liste des éléments de `L1` suivis des éléments de `L2`. Pour cela, on utilise l'opération `+`. Si par exemple `L1 = [1,2,'truc']` et `L2 = [2,'machin',2]` alors `L1 + L2` vaut `[1,2,'truc',2,'machin',2]`.
Pour concaténer plusieurs fois une liste avec elle-même, on peut utiliser l'opération `*` (avec un entier). Si par exemple `L = [2,True]` alors `L*3` vaut `[2,True,2,True,2,True]`.

Attention : Python n'effectue pas la somme des coefficients des listes. Ainsi `[2,6] + [1,3]` ne vaut pas `[3,9]` mais

Nombre d'éléments (longueur) : `len(L)` renvoie le nombre d'éléments de `L`.
Par exemple `len([2,'paf',[1,2]])` vaut

Somme des éléments : `sum(L)` renvoie la somme des éléments de `L` (si `L` contient des nombres). Par exemple `sum([4,9,-2])` vaut 11.

Appartenance :
On peut tester l'appartenance d'un élément à une liste grâce à l'opération `in`, le résultat obtenu est un booléen. Par exemple si `L = [2,3,6]` alors `2 in L` vaut `True` et `4 in L` vaut `False`.

Ajout d'un élément en fin de liste : `L.append(x)` rajoute `x` à la fin de la liste `L`.

Remarque 3
Il est aussi possible de supprimer ou de modifier des éléments d'une liste, voir plus loin.

3 Indices des éléments d'une liste

3.1 Numérotation des éléments

Rappel : on peut accéder au nombre d'éléments d'une liste `L` via l'instruction
Ainsi si `L=[2,'crac',[4,5]]` alors

Soit `L` une liste quelconque ; notons $n =$ `len(L)`. Alors les éléments de la liste `L` sont numérotés $\boxed{\text{de } 0 \text{ à } n-1}$. De plus on accède au `k`-ème élément de `L` via la syntaxe `L[k]`. Autrement dit on a :

```
L = [L[0], L[1], L[2], ..., L[n-1]]
```

Exercice 6
Que contient `x` après exécution des scripts suivants ?

```
L = [6,-5,4,2,0]
x = L[3]
```

```
L = [[3,5,2],[2,6],[1,3,0]]
y = L[1]
x = y[1]
```

Attention, si on dépasse la longueur de la liste, Python renvoie l'erreur `list index out of range`. C'est par exemple le cas avec `L=[1,0,6]` si on demande `L[ ]`.
On peut en renvanche utiliser un indice négatif. Python compte alors les éléments de la liste depuis la fin. Ainsi `L[-1]` renvoie le dernier élément de `L`, `L[-2]` l'avant dernier, etc.

3.2 Parcours des éléments par indices

On peut parcourir une liste pour consulter un à un ses éléments. On utilise pour cela une boucle `for` avec la syntaxe suivante :

```
n = len(L)
for k in range(n) :
    # instructions exploitant (ou non) l'indice k et l'element L[k]
        de la liste
```

On parle de *parcours par indices* puisque l'entier k prend pour valeurs :
c'est-à-dire les valeurs des indices des éléments de la liste.

Exercice 7
Réalisez le "suivi" des variables `k` et `a` dans la boucle for ci-dessous :

```
L = [2,5,0,3,1]
n = len(L)
a = 0
for k in range(n):
    a = a + x
```

Exercice 8
Écrire une fonction Python prenant en argument une liste de nombres réels `L` et renvoyant la somme de ses éléments.

Remarque 4
En fait, cette fonction existe déjà en Python, il s'agit de la fonction `sum`. Ainsi, `sum(L)` est la somme des éléments de `L`.

Exercice 9
Écrire une fonction Python prenant en argument une liste `L` et renvoyant son nombre d'éléments.

Remarque 5
En fait, cette fonction existe déjà en Python, il s'agit de la fonction `len`. Ainsi, `len(L)` est le nombre d'éléments de `L`.

Un algorithme classique est celui permettant d'obtenir le *minimum d'une liste* `L`.
On propose pour cela une méthode consistant à :

- initialiser une variable `mini` au premier élément de la liste qu'on obtient par
- parcourir les éléments de la liste, et actualiser la variable `mini` si on rencontre un élément plus petit.

Compléter la fonction suivante pour qu'elle renvoie le minimum d'une liste obtenu selon cette méthode :

```
def minimum(L):
    n = len(L)
    mini =
    for
        if
            mini =
    return mini
```

Quelles valeurs prennent les variables `x` et `mini` au cours de l'exécution de `minimum([4,5,1,2,3,0,-1,1])` ?

Exercice 10
Écrire une fonction `places1` prenant en argument une liste de nombres `L` et renvoyant une liste `Lbis` indiquant à quelles positions se trouvent tous les 1 que contient `L`. Autrement dit, `Lbis` contiendra l'entier k si et seulement si `L[k]` vaut 1. Par exemple, `places1([1,3,1,1,5,1])` doit renvoyer

Exercice 11
Écrire une fonction prenant en argument une liste `L` dont les éléments sont des nombres $x_0, x_1, \ldots, x_{n-1}$, et renvoyant la quantité : $\displaystyle\sum_{k=0}^{n-1} x_k\, 2^k$.

3.3 Sous-listes (slicing)

Il est possible d'accéder aux *sous-listes* d'une liste `L`, c'est-à-dire aux listes composées d'éléments consécutifs de `L`. Pour `p` et `q` deux entiers compris entre 0 et `len(L)` inclus, la syntaxe `L[p:q]` permet d'obtenir la liste des éléments de `L` compris entre les indices `p` (inclus) et `q` (exclu). En d'autres termes :

```
L[p:q] = [ L[p] , L[p+1] , L[p+2] , ... , L[q-1] ]
```

Si l'indice de départ (`p`) vaut 0, autrement dit si on considère une sous-liste qui est un *préfixe* de `L`, alors cet indice peut être omis.
De même si l'indice d'arrivée (`q`) vaut `len(L)`, autrement dit si on considère une sous-liste qui est un *suffixe* de `L`, alors cet indice peut être omis.
Lorsque les deux indices sont omis, c'est qu'on considère la sous-liste de tous les éléments de `L`.
En un mot :

```
L[:q] = L[0:q],  L[p:] = L[p:len(L)],  L[:] = L[0:len(L)] = L
```

Exercice 12
Que vaut `M` dans chacun des cas suivants ?

```python
L = [4,7,-1,0,3,2]
M = L[2:5]
```

```python
L = [4,7,-1,0,3,2]
M = L[:3]
```

```python
L = [4,7,-1,0,3,2]
M = L[4:]
```

```python
L = [4,7,-1,0,3,2]
L_bis = L[2:]
M = L_bis[:2]
```

Plus rarement, on peut avoir besoin de faire des sous-listes contenant des éléments non consécutifs dans la liste de départ. Par exemple, on peut vouloir extraire uniquement les éléments d'incides pairs. Pour cela, la syntaxe `M = L[p:q:r]` permet d'obtenir la sous-liste qu'on pourrait aussi écrire de la manière suivante :

```python
M = [L[k] for k in range(p,q,r)]
```

En d'autres termes `L[p:q:r]` contient

Exercice 13
Comment accéder à la liste des éléments d'indices pairs de `L` ?

Exercice 14
On décrit un brin d'ADN par une liste dont les éléments sont toujours un des quatre caractères `"A"`, `"T"`, `"C"` et `"G"`. Le début d'un message génétique est généralement signalé par le codon `"A","T","G"`. Écrire une fonction `depart` qui prend en argument un brin d'ADN et renvoie l'indice du début du message génétique.
Par exemple, `depart(["C","G","A","A","T","C","A","T","G","G","T","A"])` doit renvoyer `6`.
La fonction renverra un message d'erreur si le codon de départ n'est pas présent dans le message génétique considéré.

4 Modifications, suppressions

Les listes ne sont pas les seules structures en Python permettant de stocker plusieurs variables. On peut par exemple stocker n variables $x_1, \dots, x_n$ dans un n-uplet (ou *tuple* en Python) :

$$\texttt{x} = (x_1, x_2, \dots, x_n)$$

Notez la différence avec les listes :

Comme pour les listes, on accède aux éléments d'un tuple grâce à des crochets, avec une numérotation commençant à 0. Par exemple, si `x = (3,5,1,2)` alors `x[1]` vaut , `x[3]` vaut , `x[0]` vaut et `x[4]`

L'avantage des listes réside dans le fait qu'il s'agit de variables *mutables* : on peut modifier leur contenu, supprimer certains de leurs éléments et en ajouter de nouveaux (comme nous l'avons déjà fait avec la commande `append`).

4.1 Modification d'un élément d'une liste

On peut modifier le `k`-ème élément d'une liste `L`, pour lui donner la valeur `val` grâce à la commande suivante :

```
L[k] = val
```

Exercice 15
Que vaut `L` après exécution des scripts suivants ?

```
L = [4,5,2,7]
L[1] = 0
L[3] = 1
```

```
L = [4,5,2,7]
L[-2] = 0
L.append(6)
L[-2] = 1
```

Remarque 6
Ces opérations ne sont pas possibles pour les tuples : si `x=(4,5,2,7)` alors `x[1] = 0` renvoie un message d'erreur ! Les tuples, (tout comme les constantes) ne sont pas *mutables*.

Exercice 16
Écrire une fonction `transforme` qui prend en argument une liste d'entiers et la renvoie après avoir remplacé tous les `4` qu'elle contient par des `5`.

Remarque : Tout parcours d'une liste qui vise à en modifier les éléments se fait par
En effet, pour modifier un élément d'une liste, on utilise la syntaxe

4.2 Suppression d'un élément

On peut aussi supprimer un élément d'une liste. Pour cela, la commande `L.pop(k)` permet de supprimer et renvoyer le `k`-ème élément de `L`. Ainsi en écrivant :
- `L.pop(k)` on supprime l'élément `L[k]` de la liste
- `x = L.pop(k)`, on supprime l'élément `L[k]` de la liste et on le stocke dans la variable `x`

Exercice 17
Que valent `L` et `x` après exécution des scripts suivants ?

```
L = [4,5,6,3]
x = L.pop(2)
```

```
L = [4,5,6,3]
x = L.pop(0)
L.append(x)
```

```
L = [4,5,6,3]
y = L.pop(0)
x = L.pop(3)
```

Remarque 7
Lorsqu'on l'utilise sans argument, la commande `pop` agit sur le dernier élément de la liste. Ainsi `x = L.pop()` supprime le dernier élément de `L` et le place dans la variable `x`.

Remarque 8
Attention à ne pas confondre les syntaxes `L.append(x)` et `x = L.pop()`. Lorsqu'on veut rajouter un élément `x` à une liste, il est naturel de passer `x` en *argument* de la commande `append`. En revanche, lorsqu'on veut *récupérer* le dernier élément d'une liste, qui nous est inconnu, il est naturel de *définir* `x = L.pop()`.

Exercice 18
Écrire une fonction `miroir` prenant en argument une liste `L` et renvoyant la liste `L` inversée. Par exemple `miroir([1,2,3])` renverra `[3,2,1]`. On écrira deux versions de la fonction :
- `miroir1` qui videra la liste `L` pour créer sa liste inversée, et
- `miroir2` qui n'aura pas d'effet sur `L` et renverra une nouvelle liste.

5 Remarques finales

5.1 Parcours d'une liste par ses éléments

Parfois, on souhaite parcourir les éléments d'une liste, mais on n'a pas besoin de connaître leurs indices dans cette liste. Il existe pour cela un autre parcours de liste que le parcours par indices : le *parcours par éléments* dont la syntaxe est la suivante :

```python
for x in L :
    # instructions exploitant (ou non) l'element x de la liste
```

Lors de cette boucle `for`, la variable `x` prend alors successivement la valeur de tous les éléments de `L`, *de la gauche vers la droite*. Par exemple, dans la boucle `for x in [5,-2,4.3,1]`, `x` prend, dans cet ordre, les valeurs :

Remarque 9
Il n'est pas recommandé d'appeler `k` la variable de la boucle `for` comme on le fait usuellement pour une boucle `for` utilisant un `range`. En effet, ici `x` désigne un élément de `L` donc ce n'est pas nécessairement un entier !

Attention à ne pas confondre les syntaxes pour les deux parcours :

parcours par éléments :	parcours par indices :

Remarque 10

- En particulier, les syntaxes : `for k in len(L)` ou `for x in range(L)` n'ont pas de sens.
- Utilisez des noms de variables indiquant si vous faites un parcours par éléments ou par indices : il est naturel d'appeler `k` *l'indice* permettant de parcourir la liste (c'est un entier) et `x` *l'élément* de la liste (qui peut être de n'importe quel type).

Comment choisir entre un parcours par indices ou par éléments ?
Tout ce qui est fait avec un parcours par éléments peut être fait avec un parcours par indices. Mais l'inverse n'est pas vrai. Pour choisir entre les 2 parcours, il faut se demander : ai-je besoin de savoir à quel indice de la liste se situe l'élément en question ?

Exercice 19
Dans chacun des cas suivants, décider s'il faudra utiliser un parcours par indices ou par éléments :

1. Écrire une fonction prenant en argument une liste `L` contenant les nombres $x_1, x_2, \dots, x_n$ et renvoyant :

   (a) $\displaystyle\sum_{k=1}^{n} \frac{x_k}{k^2+1}$ (b) $\displaystyle\prod_{k=1}^{n} (1+3x_k)$ (c) $\displaystyle\sum_{k=1}^{n} x_k x_{n+1-k}$

2. Écrire une fonction prenant en argument une liste `L` de nombres et renvoyant la liste :
   (a) contenant tous les éléments de `L` qui sont pairs
   (b) contenant tous les éléments de `L` qui sont à une position paire

Moralité : pour éviter les erreurs, on recommande dans un premier temps de n'utiliser que des parcours par indices c'est-à-dire :

5.2 Effets de bords

Dans l'exercice précédent, on a distingué deux façons de renvoyer le mirroir d'une liste : une première fonction *modifiant* la liste, et une autre renvoyant *une autre* liste.
Lorsqu'on manipule des listes, il est en fait courant de modifier une liste prise en argument par une fonction (contrairement à ce à quoi on est habitué !). C'est le cas dès qu'on utilise sur cette liste les instructions :

Dans ce cas, on parle d'*effet de bord*, à comprendre au sens de *side effect*, c'est-à-dire effet secondaire. L'exécution de la fonction impacte la variable prise en argument de la fonction. Sauf indication contraire, on privilégiera les fonctions n'ayant pas d'effet de bords.

Exercice 20

1. Que fait la fonction suivante ?

```python
def double(L):
    for x in L:
        L.append(x)
    return L
```

2. Dans la console, un utilisateur entre les commandes suivantes :

```python
L = [7,2]
double(L)
```

Que renvoie Python ?

3. Ayant été distrait par un camarade, l'utilisateur tape à nouveau dans la console `double(L)`. Que renvoie Python ?

4. Proposer une autre façon de définir la fonction `double` n'ayant pas ce défaut.

5.3 Copie d'une liste

Pour résoudre le problème des effets de bords, on peut être tenté de copier la liste `L` prise en argument dans une liste `M` puis de travailler sur `M`. Cette solution est possible à condition de savoir comment copier une liste en Python, ce qui peut être un exercice risqué.

En effet, si `L` est une liste, alors l'instruction `M = L` ne crée pas une variable indépendante de `L`, mais donne simplement un deuxième nom à la liste `L` (contrairement à ce à quoi on est habitué !). Ainsi si on exécute par exemple les scripts suivants :

```
L1 = [4,7,1]
M1 = L1
L1[0] = 2
```

```
L2 = [4,7,1]
M2 = L2
M2.pop(1)
```

alors `L1` et `M1` valent
et `L2` et `M2` valent

Pour résoudre ce problème, il faut forcer Python à copier les éléments de `L` dans une nouvelle liste `M` en utilisant par exemple la syntaxe suivante :

```
M = L[:]
```

6 Entraînement

Exercice 21
Donner trois manières de créer une liste contenant 1000 fois le nombre 1.

Exercice 22
Écrire une fonction prenant en argument un entier `n` (supérieur à 2) et renvoyant la liste `L=`$\left[\frac{1}{2},\ \frac{1}{3},\ \frac{1}{4},\ \ldots,\ \frac{1}{n}\right]$.

Exercice 23
Soit (u_n) la suite définie par : $u_1 = 2$ et $\forall n \in \mathbb{N}^*,\ u_{n+1} = 2u_n^2 + 3$. Écrire une fonction prenant en argument n et renvoyant la liste `[`u_1`,`u_1`,`u_2`, ...,`u_n`]`.

Exercice 24
Écrire une fonction `produit` prenant en argument une liste `L` et renvoyant le produit de tous ses éléments. Par exemple `produit([2,5,3])` doit renvoyer 30.

Exercice 25
1. Écrire une fonction `maximum` prenant en argument une liste de nombres `L` et renvoyant son maximum. Par exemple `maximum([4,1,7,0])` doit renvoyer 7.
2. Écrire une fonction `ind_maximum` prenant en argument une liste de nombres `L` et renvoyant l'indice de son maximum. Par exemple `ind_maximum([4,1,7,0])` doit renvoyer 2.

Exercice 26
Écrire une fonction prenant en argument une liste de nombres réels `L` et renvoyant le nombre de fois que `L` contient `1`. Par exemple pour `L = [7,1,1,2,1,-1]` votre fonction doit renvoyer 3.

Exercice 27
Écrire une fonction `appartient` prenant en argument une liste `L` et une variable `a` et renvoyant `True` si `a` apparaît dans `L` et `False` sinon. On n'utilisera pas la fonction `in`.

Exercice 28
1. Écrire une fonction prenant en argument une liste de nombres réels `L = [`x_0`,`x_1`, ...,`x_{n-1}`]` et renvoyant la quantité $\displaystyle\prod_{k=0}^{n-1}(k + x_k)$.
2. Écrire une fonction prenant en argument une liste de nombres réels `L = [`x_1`,`x_2`, ...,`x_n`]` et renvoyant la quantité $\displaystyle\sum_{k=1}^{n}\frac{x_k}{k}$.
3. Écrire une fonction prenant en argument une liste de nombres réels `L = [`x_1`,`x_2`, ...,`x_n`]` et renvoyant la quantité $\displaystyle\prod_{k=1}^{n}\frac{x_k}{x_k^2 + 1}$.
4. Écrire une fonction prenant en argument une liste de nombres réels `L = [`x_0`,`x_1`, ...,`x_{n-1}`]` et renvoyant la quantité $\displaystyle\sum_{k=0}^{n-1}x_k x_{n-1-k}$.

Exercice 29
Écrire des fonctions Python prenant en argument une liste de nombres réels `L` et renvoyant :

1. le nombre d'éléments pairs de `L`. Par exemple pour `L = [8,1,3,2,2,1,-4]` votre fonction doit renvoyer 4 (puisque les éléments pairs de `L` sont $8, 2, 2$ et -4).
2. le nombre d'éléments pairs de `L` placés à une position paire.
   Par exemple pour `L = [8,1,3,2,2,1,-4]` votre fonction doit renvoyer 3 (puisque $8, 2, 2$ et -4 sont placés aux positions $0, 3, 4$ et 6 ; ainsi seuls le 8, le deuxième 2 et -4 sont comptés).

Exercice 30
Écrire une fonction `recherche` prenant en argument une liste `L` et un élément de type quelconque `a` et renvoyant l'indice de la première occurence de `a` dans la liste `L`. La fonction renverra `'introuvable'` si `a` n'est pas un élément de `L`.

Exercice 31
Écrire une fonction `suffixes` qui prend en argument une liste et renvoie la liste de tous ses suffixes. Par exemple, `suffixes([1,2,3])` doit renvoyer `[[1,2,3],[2,3],[3],[]]`.

Exercice 32

1. Écrire une fonction `sous_mot` prenant en arguments une liste `texte` et une liste `mot` et renvoyant `True` si `mot` est une sous-liste[1] de `texte`, et `False` sinon.
2. Si la liste `texte` est de longueur n et la liste `mot` de longueur m, combien d'opérations "élémentaires" (i.e. ici de tests d'égalités entre deux éléments des listes) réalise la fonction `sous_mot` dans le pire des cas ?

1. On entend par "sous-liste" que `L2` est toujours une sous-liste de `L1+L2+L3`, par exemple `[3,6,2]` est une sous-liste de `[5,9,3,6,2,0]`. On trouve souvent à la place de "sous-liste" le terme "facteur".