Universiteit van Stellenbosch
Departement Wiskundige Wetenskappe
Afdeling Rekenaarwetenskap

University of Stellenbosch
Department of Mathematical Sciences
Computer Science Division

Wetenskaplike Berekening 272 / Scientific Computing 272

Tutoriaal 4: Lyste in Python / Tutorial 4: Lists in Python

2020-04-30/05-07 Opgestel deur Willem Bester Gemodereer deur Brink van der Merwe

Agtergrond

Dié tutoriaal is 'n praktiese oorsig van lyste in Python. Ter voorbereiding, bestudeer die vierde reeks Python-skyfies. In u 'n oorsig van lineêere algebra—spesifiek matrikse (Vraag 7)—verlang, sien die modulewebwerf vir videolesings. Ek stel voor dat u die gids `~/wb272/tut04` skep vir u werk van dié tutoriaal.

Background

This tutorial is a practical overview of lists in Python. To prepare, study the fourth set of Python slides. If you need an overview of linear algebra—particularly matrices (Question 7)—see the module website for video lectures. I suggest you create the directory `~/wb272/tut04` for your work for this tutorial.

Uitkomste

Wanneer u die tutoriaal voltooi het, behoort u in staat te wees om die volgende te doen: (1) 'n lys te skep, (2) toegang tot lysitems te verkry en hulle by te werk deur middel van indekse, (3) gepaste funksies en metodes op lyste te roep, (4) 'n `for`-lus te gebruik om 'n lys te deurkruis, (5) 'n lys van lyste te skep, gebruik, by te werk en te deurkruis, (6) lysmodelle te skets, en (7) kort algoritmes wat op lyste staatmaak te implementeer.

Outcomes

When you have completed the tutorial, you should be able to do the following: (1) create a list, (2) access and update list items by index, (3) call appropriate functions and methods on lists, (4) use a `for` loop to traverse a list, (5) create, use, update, and traverse a list of lists, (6) draw list models, and (7) implement short algorithms that depend on lists.

Oefeninge

Exercises

1. Skep 'n lys wat die atoomgetalle van die ses alkaliese aardmetale—berillium (4), magnesium (12), kalsium (20), stronsium (38), barium (56) en radium (88)—bevat en ken dit toe aan 'n veranderlike `metals`.

   (a) By watter indeks kan Radium se atoomgetal gevind word? Kry dié getal op twee verskillende maniere, een deur middel van 'n positiewe indeks en die ander deur middel van 'n negatiewe indeks.

   (b) Watter funksie stuur terug hoeveel items daar in `metals` is?

   (c) Skryf kode wat die hoogste atoomgetal in `metals` gee. WENK: Gebruik die tabel van lysmetodes wat in die skyfies gegee word.

2. Skep 'n lys van die halfleeftye van plutonium-isotope—87.74, 24 110.0, 6 537.0, 14.4, and 376 000.0—en ken dit toe aan 'n veranderlike `half_lives`.

   (a) Skryf 'n `for`-lus wat al die waardes in `half_lives` op aparte reëls vertoon, *een waarde per reël*.

   (b) Skryf 'n `for`-lus wat al die waardes in `half_lives` *op 'n enkele reël* vertoon.

1. Create a list that contains the atomic numbers of the six alkaline earth metals—beryllium (4), magnesium (12), calcium (20), strontium (38), barium (56), and radium (88)—and assign it to a variable `metals`.

   At which index is Radium's atomic number to be found? Access this number in two different ways, one using a positive index, and the other using a negative index.

   Which function returns how many items there are in `metals`?

   Write code that gives the highest atomic number in `metals`. HINT: Use the table of list methods given in the slides.

2. Create a list of the half-lives of plutonium isotopes—87.74, 24 110.0, 6 537.0, 14.4, and 376 000.0—and assign it to a variable `half_lives`.

   Write a `for` loop that displays all the values in `half_lives` on separate lines, *one value per line*.

   Write a `for` loop that displays all the values in `half_lives` *on a single line*.

3. Skep 'n lys van temperature (in grade Celsius) met die waardes 25.2, 16.8, 31.4, 23.9, 28, 22.5 en 19.6, en ken dit toe aan 'n veranderlike `temperatures`. Doen nou die volgende in die Python-interpreteerder:

   Create a list of temperatures (in degrees Celsius) with the values 25.2, 16.8, 31.4, 23.9, 28, 22.5, and 19.6, and assign it to a variable named `temperatures`. Now, do the following in the Python interpreter:

   (a) Gebruik een van die lysmetodes om `temperatures` in toenemende volgorde te sorteer.

   Use one of the list methods to sort `temperatures` in ascending order.

   (b) Gebruik snitte en skep twee nuwe lyste, `cool` en `warm`, wat die temperature onder en bo 20°C, onderskeidelik, bevat. Gebruik die gesorteerde lys as beginpunt. U mag die snypunte hardkodeer.

   Use slicing, and create two new lists, `cool` and `warm`, that contain the temperatures below and above 20°C, respectively. Use the sorted list as starting point. You may hard-code the slicing indices.

   (c) Gebruik lysrekenkunde en herkombineer `cool` en `warm` tot 'n nuwe lys genaamd `temps_celsius`.

   Use list arithmetic, and recombine `cool` and `warm` into a new list named `temps_celsius`.

   (d) Skryf 'n `for`-lus om al die waardes van die lys `temps_celsius` tot grade Fahrenheit om te skakel en stoor die omgeskakelde waardes in 'n nuwe lys `temps_fahrenheit`. Die oorspronlike lys moet onveranderd bly.

   Write a `for` loop to convert all the values from the list `temps_celsius` into degrees Fahrenheit, and store the converted values in a new list named `temps_fahrenheit`. The original list must remain unchanged.

   (e) Skep 'n *kopie* van die lys `temps_celsius` en ken dit toe aan 'n veranderlike genaamd `temps`; pas op vir aliasering. Skakel dan dié temperature in grade Celsius *in situ* na grade Fahrenheit om; dit wil sê, die lys `temps` word bygewerk tydens verwerking.

   Create a *copy* of the list `temps_celsius` and assign it to a variable named `temps`; watch out for aliasing. Then convert these temperatures in degrees Celsius to degrees Fahrenheit in-place; that is, update the list `temps` during processing.

4. Skep 'n **geneste lys**—met ander woorde, 'n lys van lyste—waar elke item in die buitenste lys 'n lys is wat die elementnaam, atoomgetal en atoomgewig vir 'n alkaliese aardmetaal bevat. Die waardes is berillium (4 en 9.012), magnesium (12 en 24.305), kalsium (20 en 40.078), stronsium (38 en 87.62), barium (56 en 137.327) en radium (88 en 226). Ken die lys toe aan 'n veranderlike genaamd `metals`.

   Create a **nested list**—that is, a list of lists—where each item in the outer list is a list that contains the element name, atomic number, and atomic weight for an alkaline earth metal. The values are beryllium (4 and 9.012), magnesium (12 and 24.305), calcium (20 and 40.078), strontium (38 and 87.62), barium (56 and 137.327), and radium (88 and 226). Assign the list to a variable named `metals`.

   (a) Skryf 'n `for`-lus om die elementnaam, atoomgetal en atoomgewig vir elke alkaliese aardmetaal te vertoon, waar die data vir elke metaal op 'n aparte reël verskyn, geformatteer soos hieronder geïllustreer.

   Write a `for` loop to display the element name, atomic number, and for each alkaline earth metal, where the data for each metal appears on a separate line, formatted as illustrated below.

   ```
   beryllium:  4   9.012
   magnesium: 12  24.305
   calcium  : 20  40.078
   strontium: 38  87.620
   barium   : 56 137.327
   radium   : 88 266.000
   ```

   (b) Gebruik *geneste* `for`-lusse om 'n nuwe lys genaamd `data` te skep wat die data in `metals` in die oorspronklike volgorde bevat, maar nie genes nie. (Dit word die **platmaak** van 'n geneste lys genoem.)

   Use *nested* `for` loops to create a new list named `data`, which contains the data in `metals` in the original order, but not nested. (This is called **flattening** a nested list.)

5. Die volgende funksie bevat nie 'n dokstring of kommentaar nie. Skryf genoeg van albei om die maklik te maak vir iemand om te verstaan wat die funksie doen en hoe die funksie dit doen. Vergelyk dan u antwoord met dié van ten minste twee ander mense. Hoe soortgelyk is hulle? Waarom verskil hulle?

   The following function does not have a docstring or comments. Write enough of both to make it easy for someone else to understand what this function does and how the function does it. Then compare your solution with those of at least two other people. How similar are they? Why do they differ?

   ```
   def mystery_function(values):
   ```

```
    result = []
    for i in range(len(values[0])):
        result.append([values[0][i]])
        for j in range(1, len(values)):
            result[-1].append(values[j][i])
    return result
```

6. Skets 'n geheuemodel wat die effek van die volgende stellings toon:

   Draw a memory model showing the effect of the following statements:

   ```
   values = [0, 1, 2]
   values[1] = values
   ```

7. Skryf 'n Python-funksie `matrix_transpose` wat 'n matriks transponeer. Die funksie neem 'n lys van lyste, wat die matriks in ryhooforde voorstel, as parameter. U mag aanneem dat die geneste lyste reghoekig is; dit wil sê, al die lyste genes in die buitenste lys het dieselfde lengte. Vir 'n $(m \times n)$-matriks $A = [a_{ij}]$, waar $1 \leq i \leq m$ en $1 \leq j \leq n$, word die getransponeerde matriks gedefinieer as $A^T = [a_{ji}]$; let op die omruiling van die onderskrifte. Stuur 'n lys van lyste, wat die getransponeerde matriks A^T in ryhooforde voorstel, terug. Stoor u funksie in 'n lêer genaamd `mat.py` en gebruik die program `test_mat.py`, beskikbaar in 'n argief op die webblad, om u werk te toets. ONTHOU: Python se lysindekse begin by nul.

   Write a Python function `matrix_transpose` that transposes a matrix. The function takes a list of lists, representing the matrix in row-major order, as parameter. You may assume that the nested lists are rectangular; that is, all lists nested in the outer list are of equal length. For an $(m \times n)$ matrix $A = [a_{ij}]$, where $1 \leq i \leq m$ and $1 \leq j \leq n$, the transpose is defined as $A^T = [a_{ji}]$; note the reversal of subscripts. Return a list of lists, representing the transpose A^T in row-major order. Save your function to a file named `mat.py` and use the program `test_mat.py`, available in an archive on the web page, for testing. REMEMBER: Python starts list indices at zero.

8. Skryf 'n Python-funksie `closest` wat 'n lys van (x, y)-koördinate in twee-dimensionele Euklidiese ruimte as parameter ontvang en besluit watter van die punte die naaste (in die Euklidiese sin) aan die oorsprong $(0, 0)$ is. Die koördinate word as 'n lys van talle aangestuur. Die funksie moet 'n drietal (p, i, d) terugstuur, waar p die punt die naaste aan die oorsprong, i sy indeks in die lys van talle en d sy afstand van die oorsprong af is. U mag nie `if`-stellings gebruik nie, maar neem aan dat geen twee punte op dieselfde afstand naaste aan die oorsprong lê nie. Stoor u funksie in 'n lêer genaamd `distance.py` en gebruik die program `test_distance.py`, beskikbaar in 'n argief op die webblad, om u werk te toets. WENK: Gebruik standaard `list`-metodes, asook die ingeboude `min`-funksie.

   Write a Python function `closest` that receives as parameter a list of (x, y)-coordinates in two-dimensional Euclidean space and decides which of these points is closest (in the Euclidean sense) to the origin $(0, 0)$. The coordinates are passed as a list of tuples. The function must return a tuple (p, i, d), where p is the point closest to the origin, i is its index in the list of tuples, and d is its distance from the origin. You may not use `if` statements, but assume that no two points lie at the same distance closest to the origin. Save your function to a file named `distance.py` and use the program `test_distance.py`, available in an archive on the web page, for testing. HINTS: Use standard `list` methods, as well as the built-in `min` function.

9. 'n Agtsyfer studente- of personeelnommer by die Universiteit van Stellenbosch is geldig as en slegs as die volgende vergelyking waar is:

   An eight-digit student or staff number at Stellenbosch University is valid if and only if the following equation holds true:

$$\left(\sum_{k=1}^{8} k d_k\right) \bmod 11 = 0, \tag{1}$$

   waar k die indeks van 'n syfer d_k is wanneer die syfers van *regs na links* genommer word, vir $1 \leq k \leq 8$. Dit wil sê, elke syfer word met sy indeks vermenigvuldig, dié produkte word bymekaar getel en die som modulo 11 moet gelyk wees aan nul.

   where k is the index of a digit d_k when the digits are numbered from *right to left*, for $1 \leq k \leq 8$. That is, each digit is multiplied by its index, these products are added, and the sum modulo 11 must be equal to zero.

VOLTI SUBITO

VOLTI SUBITO

Skryf 'n Python-funksie `is_valid_usnum` wat een heelgetal as parameter neem en `True` of `False` terugstuur, afhangende van of die parameter 'n geldige US-nommer is al dan nie. U moet 'n `for`-lus gebruik om die onderskeie syfers te "isoleer". U mag aanneem die parameter is 'n nie-negatiewe heelgetal wat uit presies agt syfers bestaan. `Wenk: True` en `False` is die enigste twee waardes van die ingeboude boolese tipe. Alhoewel ons nog nie `if`-stellings gedek het nie, kan ons steeds maklik `True` of `False` terugstuur: Bereken die linkerkant van Vgl. (1), stoor dié resultaat in 'n veranderlike, sê, `checksum`, en gebruik die volgende as laaste stelling in u funksie: `return checksum == 0`

Write a Python function `is_valid_usnum` that takes one integer as parameter, and returns `True` or `False`, depending on whether or not the parameter is a valid US number. You must use a `for` loop to "isolate" the individual digits. You may assume that the parameter is a non-negative integer which has exactly eight digits. HINT: `True` and `False` are the only values of the built-in boolean type. Although we have not yet covered `if` statements, we can still easily return `True` or `False`: Calculate the left-hand side of Eq. (1), store this result in a variable, say, `checksum`, and then write the following to return from your function: `return checksum == 0`

10. Toets u funksie van Vraag 9 deur 'n Python-program te skryf wat die US-nommer om te toets as 'n bevelreëlargument inlees. U mag aanvaar dat so 'n bevelreëlargument uit presies agt karakters sal bestaan. Stoor u program in 'n lêer genaamd `testnum.py`. Voorbeelde van hoe die program geloop word, word hieronder gegee. WENKE: Onthou dat bevelreëlargumente as stringe na Python aangestuur word, maar dat u funksie 'n heelgetal as parameter neem. U kan die funksie se terugstuurwaarde vertoon deur bloot 'n `print`-bevel vooraan die funksieroep te plaas.

Test your function of Question 9 by writing a Python program that reads the US number to test as a command-line argument. You may assume such a command-line argument to consist of exactly eight characters. Save your program to a file called `testnum.py`. Examples of how the program is run are given below. HINTS: Remember that command-line arguments are passed to Python as strings, but that your function takes an integer as parameter. You can display the return value of the function by prefacing the function call with a `print` command.

```
12345679@ibm-lab25:~/wb272/tut04$ ./testnum.py 11223344
False
12345679@ibm-lab25:~/wb272/tut04$ ./testnum.py 44332211
True
12345679@ibm-lab25:~/wb272/tut04$ ./testnum.py 12345679
True
12345679@ibm-lab25:~/wb272/tut04$ ./testnum.py 12345678
False
```

11. Skryf nog 'n weergawe van die `is_valid_usnum`-funksie van Vraag 9 wat, in plaas van 'n heelgetal, die US-nommer as 'n stringparameter neem. U nuwe weergawe moenie die hele string op een slag na 'n heelgetal omskakel nie, maar moet eerder die individuele karakters een vir een met behulp van 'n `for`-lus onttrek en elkeen apart as 'n syfer vertolk.

Write another version of the `is_valid_usnum` function of Question 9 that, instead of an integer, takes the US number as a string parameter. Your new version must not convert the entire string to an integer in one go, but must instead isolate the individual characters one by one in a `for` loop and interpret each separately as a digit.

12. 'n US-nommer bestaan in der waarheid slegs uit die eerste sewe syfers (van links na regs). Die agtste syfer—met ander woorde, d_1—is eintlik 'n kontrolesyfer wat met behulp van Vgl. (1) bereken word. Kontrolesyfers is 'n tipe oortolligheidstoets wat help om transkripsiefoute op te spoor en ook 'n mate van sekuriteit verskaf—bankkaarte en Suid-Afrikaanse ID's is ander voorbeelde wat sulke kontrolesyfers gebruik. Skryf nou 'n Python-funksie wat 'n sewesyfer, nie-negatiewe heelgetal as parameter neem en dan die kontrolesyfer met behulp van Vgl. (1) bereken. Dit wil sê, u funksie ontvang die syfers $d_8 d_7 \cdots d_2$ en moet d_1 terugstuur. Die volgende definisies mag nuttig wees:

In reality, a US number consists only of the first seven digits (from left to right). The eight digit—that is, d_1—is actually a check digit that is calculated from Eq. (1). A control digit is a kind of redundancy check which helps to detect transcription errors and also provides a measure of security—bank cards and South African IDs are other examples that use control digits. Now, write a Python function that takes a seven-digit, non-negative integer as parameter and then calculates the control digit from Eq. (1). That is, your function receives the digits $d_8 d_7 \cdots d_2$ and must return d_1. The following definitions may be useful:

$$(a \bmod n) + (b \bmod n) = (a + b) \bmod n, \tag{2}$$

$$(a \bmod n)(b \bmod n) = ab \bmod n. \tag{3}$$