Chapter 2: Lists and for loops

Author: Vincent Delecroix

In the first tutorial, you learnt how to call Sage functions and solve basic problems. To do more advanced computations you will need to learn a bit of programming. The aim of this tutorial is to introduce you to some basic Python programming: `list` that is one of the builtin Python data structures and `for` loops.

For reference you can have a look at [pydoc-datastructures] and [pydoc-controlflow].

Manipulating lists

In Python lists are constructed using square brackets (`[` and `]`) such as:

```
sage: l = [1, "two", -5.7, Primes()]
```

As you can notice, the data contained in a list is not necessarily homogeneous. To access elements of a list, one also uses square brackets:

```
sage: l[0]    # the first element
sage: l[2]    # the third elements
sage: l[-1]   # the last element
sage: l[4]    # this is out of range
```

An important function is `len` that returns the length of the list:

```
sage: len(l)   # number of elements in l
```

Given a list `l` and a non-negative integer `n` the operation `l * n` duplicates `n` times the list `l`:

```
sage: l * 3
```

Given two lists `l1` and `l2` the operation `l1 + l2` is the concatenation of `l1` and `l2`:

```
sage: [1, 2, 3] + [4, 5, 6]
```

Exercise 2.1

Construct the list that contains 1 time 1, 2 times 2, 3 times 3, ..., 10 times 10 (i.e. `[1, 2, 2, 3, 3, 3, ..., 10]`

A for loop is a control flow that will allow to repeat some instructions for all elements of an *iterable* such as list.:

```
sage: l = [1, "hello", 3]
sage: for x in l:
....:     print(x)
```

You can have more than one instructions inside a for loop:

```
sage: l = [1, 2, 3, 4, 5]
sage: for x in l:
....:     y = 3 * x^ - 2
....:     print(y)
```

And it is possible to nest them:

```
sage: l1 = [1, 2, 3]
sage: l2 = [4, 7, 13]
sage: for x in l1:
....:     for y in l2:
....:         print(x, y, x^y)
```

Often you might want to perform a for loop for integers in a given range. To do so there are the functions `range` (from Python) and `srange` (from Sage).:

```
sage: for i in range(10):
....:     print(i)
```

```
sage: for i in srange(10):
....:     print(i)
```

The difference between `range` and `srange` is that `range` produces Python integer while `srange` produces Sage integers. You can compare:

```
sage: for i in range(10):
....:     print(i.is_prime())
```

```
sage: for i in srange(10):
....:     print(i.is_prime())
```

For both `range` and `srange` there are three possible syntaxes

- $range(stop)$: integers from 0 to $stop-1$ included
- $range(start, stop)$: integers from $start$ to $stop-1$
- $range(start, stop, step)$: integers from $start$ to $stop$ and with a difference of $step$ between each consecutive terms

Exercise 2.2

Write a for loop which prints the factorization for each integer from 1 to 100.

Exercise 2.3

Write a for loop which prints the primitive of the functions $\sin(x)$, $\cos(x)$, $\tan(x)$, $\log(x)$, $\exp(x)$, $\sinh(x)$ and $\cosh(x)$.

Write a for loop which prints the derivates of the functions $f_n(x) = x^n$ for all values of n from 1 to 20.

Exercise 2.4

Redo exercise 2.1 using a `for` loop

Repeat it with 100 instead of 10

Exercise 2.5

Write some code that displays the following figure using a loop

```
sage: print('*')
....: print('**')
....: print('***')
....: print('****')
....: print('*****')
....: print('******')
```

Write some code that displays the following figure using a loop:

```
sage: print('*')
....: print('**')
....: print('***')
....: print('****')
....: print('*****')
....: print('******')
....: print('*****')
....: print('****')
....: print('***')
....: print('**')
....: print('*')
```

Write some code that displays the following figure using a loop:

```
sage: print('   *')
....: print('  ***')
....: print(' *****')
....: print('*******')
....: print(' *****')
....: print('  ***')
....: print('   *')
```

Exercise 2.6

What is the value of

$$\sum_{k=1}^{20} k^k$$

Solve $Euler problem 48 < https://projecteuler.net/problem = 48 >$.

Note that Python `tuple` (constructed with parenthesis) and Python `string` (constructed with quotes `'` or `"`) behave the same with respect to many operations:

```
sage: t1 = (0, "hello", -2/3)

sage: t1[0]

sage: t1[-1]

sage: t1 * 3

sage: s1 = "I am in"
sage: s2 = "Saint-Flour"
sage: s1 + " " + s2
```

Exercise 2.7

Here is a list describing the size of 64 files:

```
sage: L = ["12K", "12K", "12K", "12K", "12K", "12K", "12K", "12K", "16K",
....: "16K", "16K", "20K", "20K", "20K", "24K", "24K", "24K", "24K", "24K",
....: "24K", "24K", "24K", "24K", "28K", "32K", "40K", "4K", "4K", "4K", "4K",
....: "4K", "4K", "4K", "4K", "4K", "4K", "4K", "4K", "4K", "4K", "4K", "4K",
....: "4K", "4K", "4K", "4K", "4K", "4K", "4K", "4K", "4K", "4K", "4K", "4K",
....: "4K", "4K", "4K", "4K", "8K", "8K", "8K", "8K", "8K", "8K"]
```

Compute the total size of the files, their average and the median

(*hint*: to convert a string `s` like `"12"` into an integer simply do `ZZ(s)`)

How many files are there of each size? (you might want to look the available method on list object)

Construct the string that is the concatenation of each file size separated by a space

Can you construct the same string by increasing size?

(*hint*: you might want to sort the list first... use tab-completion to find the appropriate list method to do this)

And by decreasing size?

(*hint*: have a look at the documentation of the method you used for sorting in the previous question)

Exercise 2.8

Let S_0 be the unit square with vertices $(0,0)$, $(1,0)$, $(1,1)$ and $(0,1)$. We define the square S_n obtained by joining the middle of each side of S_{n-1}.

Draw on the same pictures S_0, S_1, S_2, ... up to S_{10}.

Do the same graphics starting from another quadrilateral which is not regular

Do the same starting from a pentagon (five vertices)

Exercise 2.9

What does the following code?:

```
sage: x = 1.0
sage: for i in range(10):
....:     x = (x + 2.0 / x) / 2.0
```

What is the difference with:

```
sage: x = 1
sage: for i in range(10):
....:     x = (x + 2 / x) / 2
```

Exercise 2.10

Solve Euler problem 13. To simplify the exercise, the list of numbers has been created in the following cell:

```
sage: L = [37107287533902102798797998220837590246510135740250,
....: 46376937677490009712648124896970078050417018260538,
....: 74324986199524741059474233309513058123726617309629,
....: 91942213363574161572522430563301811072406154908250,
....: 23067588207539346171171980310421047513778063246676,
```

```
....: 892616706966236338201363784183836841787343617267 57,
....: 281128798128499794080654819315926216912758898327 38,
....: 442742289174325203219235894228767964876702721893 18,
....: 474514457360013064390911672168568445887116031532 76,
....: 703864861058430254399396198289175936656867579349 51,
....: 621764571418565606295021572231965867550793241933 31,
....: 649063524627419049291014324458138226633479447581 78,
....: 925758677183372176619637515905792397282455988384 07,
....: 582035653253593990084026335689488301894586282278 28,
....: 801811993848262820142781941399405675871511700943 90,
....: 353986643728271126538299872407844730531901042935 86,
....: 865155060062958648615320752733719591914205172558 29,
....: 716938887077154664991155934876035329217149700569 38,
....: 543700705768266846246214956500764717872944383776 04,
....: 532826541087568284431911906346940378552177792951 45,
....: 361232725250002960710750825638156567108852583507 21,
....: 458765761724109764473391106072182652368772236360 45,
....: 174237069058518606604482076212098132878607339694 12,
....: 811426604180868306193284608111910615569405126896 92,
....: 519343254517283886419180470492932150586425630494 83,
....: 624672216484350762017279180399446930047329563406 91,
....: 157324443869081257945140890577062294291971079282 09,
....: 550376875256787730918625407449698445083303936821 26,
....: 183363848253301546861961243487676812975343759465 15,
....: 803862875928784902015216855548287172012192577669 54,
....: 781828337579931036147403568564490955270978647975 81,
....: 167263201004368978425535399209318374414978068609 84,
....: 484030981290777917990882187953273644756755908480 30,
....: 870869875513927118545170785441618524243206931503 32,
....: 599594068957565367821070749269665376763262354472 10,
....: 697939506796526947425977097391666937630426339870 85,
....: 410526847082990852113994273657341161827603150012 71,
....: 653786073615010808570091499395125570281987460043 75,
....: 358290353174347173269321235781549826297425527373 07,
....: 949537597651053059469660676831565743771674018752 75,
....: 889028025717332296191766687138199318110487701902 71,
....: 252676802760780030136786809925254634010616328665 26,
....: 362702185404977055856299465806362379931407462559 62,
....: 240744869082311749777923654662572469233228109171 41,
....: 914302881971032885978066697608929386382850253334 03,
....: 344130655780161278159218150055618688364684200904 70,
....: 230530811728164304876237919698424872550366387845 83,
....: 114876969321549028104240201383351244621814417734 70,
....: 637832994906362596664985876182212252255124867645 33,
....: 677201869716985443124195724099139590089523100588 22,
....: 955482553002635207815322967962494816419538682187 74,
....: 760853271322857231104248034561248676970645079952 36,
....: 377742425354112916842768655389262050249103265729 67,
....: 237019132757256752856532482582654630922070585965 22,
....: 297988602722583319131263751473419948895347657455 01,
....: 184957014548792889848568277260777137214037988797 15,
....: 382982037830314735277215803481445134913732266513 81,
....: 348295438291999181802789165224310273922511228695 39,
....: 409579530664052326325380441000596549391598795936 35,
....: 297461521855023713076422551211836938035803885849 03,
....: 416981162220729771861582366784246891579935329619 22,
```

```
....: 624679571944012690438771072750481023908955235974 57,
....: 231897067725479150615055049539229795309011299675 19,
....: 861880882258753145295840992512038290094077707756 72,
....: 113067397083047244838165338735023408456470580773 08,
....: 829591747671403631980081871290118754913105471265 81,
....: 976233310448183862695154563349263665728975634005 00,
....: 428462801835170705278318394258821455212272512503 27,
....: 551216035469812005817621652128276527516912968977 89,
....: 322381957343293399464375019078369457658833523998 86,
....: 755061649651847751807381688378610915273579297013 37,
....: 621778427521926234019423996391680449839931733127 31,
....: 329241857071473495669166746876346609150359146775 04,
....: 995186714302352196288948901024233251169136196266 22,
....: 732674608005915474718307983928685352069469445407 24,
....: 768418225246744171615140364279822733480555562148 18,
....: 971426179103425986472045168939894221798260880768 52,
....: 877836461827993463137677543078093633330189826420 90,
....: 108488025216746708832151201858835432238128769527 86,
....: 713296124747824645386369930090493103636197638780 39,
....: 621840735723997942234062353938083396513274080111 16,
....: 666278919814880877979418768761442300309844908514 11,
....: 606618262936828367647447792391803351109890697907 14,
....: 857869440895529906536404474255760836599766457950 96,
....: 660243964099053896071201982199760475994901972302 97,
....: 649139826800329731560371200413779037855660850892 52,
....: 167309393198727502754689069037075394130426523150 11,
....: 948093772450487951509541009216458637547105984367 91,
....: 786391670211874924319957006419179697775990283006 99,
....: 153687137119366149528113058763802784107544497330 78,
....: 407899231155355625611423224232550336854424889173 53,
....: 448899115014406480203690680639606723221932041495 35,
....: 415031288803395360532993403680069777106505666319 54,
....: 812348806732101467390585685579345814036278227032 80,
....: 826165707739483275922328459417065250945123252306 08,
....: 229188020587773197198394501808880724296619808111 97,
....: 771585425020165450904132458097868827789487218596 17,
....: 721078384350691861554356628840622574736922845095 16,
....: 208496039801340017239306716668235552452528046097 22,
....: 535035342264725242508740540755917897812643303316 90]
```

Modifying lists

Here are three ways to modify an already existing list `l`:

- `l[i] = j`: modify the element at position `i` to become `j`
- `l.append(j)`: append the element `j` at the end of `l`
- `l.pop()`: remove the last element of the list `l` and return it
- `l.extend(ll)`: add to `l` the content of the iterable `ll`

(there is also `l.insert(i, j)` that we will not use)

Exercise 2.11

What is the value of the list `l` at the end of the execution:

```
sage: l = [1, 2, 3]
sage: l.append(-1)
sage: l[1] = 7
```

(think about it before executing the lines)

What is the value of the list `l` at the end of the execution:

```
sage: l = [1]
sage: l.extend(l)
sage: l.extend(l)
```

(think about it before executing the lines)

Exercise 2.12

Given the following list of integers:

```
sage: l = [231, 442, 534, 667, 827, 314, 299, 351, 257, 688, 661, 123,
....: 567, 247, 151, 222, 605, 307]
```

Modify it so that each number at an even position is replaced by its double

Exercise 2.13

Let the following lists:

```
sage: t1 = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
sage: t2 = ['January', 'February', 'March', 'April', 'May', 'June',
....: 'July', 'August', 'September', 'October', 'November', 'December']
```

Using `t1` and `t2` create a new list `t3` containing all elements of the two lists alternating them in such a way that each Month is followed by the corresponding number of days, that is, `['Janvier',31,'Février',28,'Mars',31, etc...]`

Exercise 2.14

Using the recurrence relation satisfied by the binomial numbers

$$\binom{n+1}{k} = \binom{n}{k} + \binom{n}{k-1}$$

compute the list $\binom{20}{0}, \binom{20}{1}, \ldots, \binom{20}{20}$. In order to do that you need to start from the list $[1]$ and design a loop that constructs successively $[1,1]$, then $[1,2,1]$, then $[1,3,3,1]$, etc. (It can be done using only one list.)

Modify your loop to compute the Stirling numbers of the second kind that satisfies

$$S(n+1,k) = kS(n,k) + S(n,k-1)$$

with initial conditions $S(0,0) = 1$ and $S(n,0) = S(0,n) = 1$.

List comprehension

List comprehension is a flexible way to build list. To build the list of squares n^2 from $n = 1$ to $n = 10$ one can do:

```
sage: l = [n^2 for n in srange(1, 11)]
....: print(l)
```

Exercise 2.15

Construct the same list of squares using a *for* loop and the method *.append*().

Exercise 2.16

Construct the list of powers of 5 for all values of exponents in the interval $[1,20]$

Exercise 2.17

The Fibonacci sequence is defined by $F_0 = 0$, $F_1 = 1$ and for all $n \geq 2$, $F_n = F_{n-1} + F_{n-2}$.

Make the list of the first 50 Fibonacci numbers F_n.

Using a for loop, print the values of $F_n^2 - F_{n-1}F_{n+1}$ for n between 1 and 48.

Using a for loop, print the values of $F_{2n} - F_n^2 - F_{n-1}^2$.

What do you remark? Could you prove it?

Exercise 2.18

What happens when you try to modify a string or a tuple?

Exercise 2.19

Solve the Euler problem 18 and 67 about "maximum sum paths"

Exercise 2.20 (Champernowne constant)

The Champernowne constant is the real number obtained by concatenating all natural numbers in base 10

$$C = 0.12345678910111213141516171819202122 23\ldots$$

The *Champernowne word* is the sequence of digits. Using a for loop, construct the begining of the Champernowne word obtained by concatenating the integers from 1 to 100

(*hint*: use the method `n.digits()` of Sage integers together with the method `l.extend(ll)` on lists)

Then solve Euler problem 48

Further comments

To learn more about iterators (that can be thought as "lazy lists") and in particular how to construct them, you can have a look at the Sage thematic tutorial on Comprehensions [sagett-comprehensions].

References

[pydoc-datastructures] https://docs.python.org/2.7/tutorial/datastructures.html

[pydoc-controlflow] https://docs.python.org/2.7/tutorial/controlflow.html

[sagett-comprehensions] https://doc.sagemath.org/html/en/thematic_tutorials/tutorial-comprehensions.html