Computer Programming I

Lecture 3a: Lists and Strings

Johan Öfverstedt

Lists

We have seen in previous lectures and labs that lists stores / represents a sequence of objects / values, each being addressed by an integer index:

xs: 0, 1, ..., len(xs)-1 -> elem_0, elem_1, ..., elem_(len(xs)-1)

Index	0	1	2	3	4
Värde	'a'	'B'	'c'	'D'	'e'

Slicing - Manipulation of sublists

Slicing is an operation that extracts a subset of a list in a systematic way, to enable reading or changing the list in place.

```
xs = [1, 2, 3, 4, 5, 6, 7]
print(xs[1:-1:2])
>> [2, 4, 6]
print(xs[1:-2:2])
>> [2, 4]
xs[1::2] = [13, 15, 17]
print(xs)
>> [1, 13, 3, 15, 5, 17, 7]
```

Slicing can be used to pick out sublists (with start:stop:step notation)

and you can also assign to the sublist to change the existing list in place

Negative numbers for start/stop index means indexing from the end of the list:
-n -> len(xs)-n

Slicing - Manipulation of sublists

We can obtain a slicing in reverse order by providing a negative step, and a start index that is larger than stop:

```
xs = [1, 2, 3, 4, 5, 6, 7]
print(xs[5:2:-1])
>> [6, 5, 4]
```

Slicings, just like range(start, stop, step),
Uses half-open intevals:
[start, stop)
[2:5] -> index [2, 3, 4]

Slicing - Manipulation of sublists

Since -1 refers to the last element and -2 the penultimate element etc (which helps so you don't have to write len(xs)-1, len(xs)-2, ..., etc) it is easy to make hidden mistakes that don't give an error message

```
xs = [1, 2, 3, 4, 5, 6, 7]
ys = []
for i in range(len(xs)):
    xss = xs[i-1:i+2]
    ys.append(sum(xss)/3.0)
print(ys)
>> [0.0, 2.0, 3.0, 4.0, 5.0, 6.0, 4.333333333333333]
```

Slicing can be used to pick out sublists (with start:stop:step notation)

and you can also assign to the sublist to change the existing list in place

Logical error. Based on the sum of an empty slicing

Slicing - Indexing out of bounds

If you slice a list in a way that the boundaries are out of bounds, you do not get an error which you do get if you index out of bounds for a single element:

```
x = [1, 2, 3]
x[5]
>> Traceback (most recent call last):
>>   File "<stdin>", line 1, in <module>
>> IndexError: list index out of range
x[4:5]
>> []
```

List Building

Lists in Python can be constructed in many different ways:

- Enumeration ([])
- Concatenation (+)
- Repetition (*)
- Insertion (insert/append)
- Generator
- List comprehension

Enumeration

You may create a list by enumerating the elements one by one in the code:

```
xs = [1, 2, 3, 4, 5]
print(xs)
>> [1, 2, 3, 4, 5]
```

If you are able to write down all the elements of the list, especially if they do not follow a simple mechanical rule, it may be easiest to enumerate them.

For this specific example, you can just do this:

```
xs = list(range(1, 6))
```

Concatenation

Given two lists, you can concatenate them together to form a longer list containing the contents of both:

```
xs1 = [1, 2, 3, 4, 5]
xs2 = [6, 7, 8, 9, 10]

xs3 = xs1 + xs2

print(xs3)
>> [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Repetition

Given a list you can repeat the contents of the list a number of times:

```
xs1 = [1, 2, 3, 4, 5]
xs2 = xs1 * 3

print(xs2)
>> [1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
```

Insertion with insert/append

You can construct a list element by element by adding them into a list with append and insert:

`list.append(value)`: adds a new element last (and grows the list by one)
`list.insert(index, value)`: adds a new element at a given index, grows the list by one, and displaces all the following elements.

Insertion with insert/append

You can construct a list element by element by adding them into a list with append and insert:

```
xs1 = []
xs1.append(3)
xs1.append(4)
xs1.append(5)
xs1.insert(0, 1) # Lägger till 1 på index 0
xs1.insert(1, 2) # Lägger till 2 på index 1

print(xs1)
>> [1, 2, 3, 4, 5]
```

Generator

You can construct a list by making a generator, and then converting that into a list:

```
def sq_gen(n):
    for i in range(1, n+1):
        yield i*i

xs = list(sq_gen(5))
print(xs)
>> [1, 4, 9, 16, 25]
```

List comprehension

You can construct a list in a very compact, and elegant way with the list comprehension syntax:

```
xs = [x for x in range(1, 6)]
print(xs)
>> [1, 2, 3, 4, 5]

xs = [x*x for x in range(1, 6)]
print(xs)
>> [1, 4, 9, 16, 25]
```

List comprehension

You can produce multiple values at the same time with tuples

```
xs = [(x, x*x) for x in range(1, 6)]
print(xs)
>> [(1, 1), (2, 4), (3, 9), (4, 16), (5, 25)]
```

Or with lists (to form a list of lists):

```
xs = [[x, x*x] for x in range(1, 6)]
print(xs)
>> [[1, 1], [2, 4], [3, 9], [4, 16], [5, 25]]
```

Elementary algorithms for lists

Find - linear search - finds the index of a sought value

```
def find(haystack, needle):
    for i in range(0, len(haystack)):
        if haystack[i] == needle:
            return i
    return None
```

Built-in functions in Python

`haystack.index(needle)`

(gives ValueError if the needle is not found)

Requires `len(haystack)` iterations and comparisons in the worst-case when the needle is not found.

EXTRACURRICULAR

Elementary algorithms for lists

Find - binary search - finds the index of a sought element in a ***sorted*** list.

```
def find_sorted(haystack, needle):
    i1 = 0
    i2 = len(haystack)
    while i1 < i2:
        mid_ind = i1 + (i2-i1)//2
        if haystack[mid_ind] == needle:
            return mid_ind
        elif needle < haystack[mid_ind]:
            i2 = mid_ind
        else:
            i1 = mid_ind + 1
    return None
```

Requires log_2(`len(haystack)`) iterations and comparisons [32 comparisons for ~4 billion elements in the list.]

log_2 is the base 2 logarithm.

Elementary algorithms for lists

Minimum (Maximum, change < into >)

```
def minimum(xs):
    assert(len(xs)>0)
    ind = 0
    value = xs[0]
    for i in range(1, len(xs)):
        if xs[i] < value:
            value = xs[i]
            ind = i
    return (value, ind)
```

Built-in function in Python
`min(xs)` gives smallest value,
`index(min(xs))` gives the index of the smallest value

Programming with lists

Much of programming can be summarized with:
Interaction with the hardware, graphics, input/output, networking, etc.

AND

List construction - Creation of lists from algorithms or existing data.
List transformations sorting conversion, arithmetic
Reduction of lists (minimum/maximum, summation, etc)

Python has great features that makes it easy to manipulate lists, such that all those steps are easy and succinct, requiring little code.

Strings

Strings are sequences of characters. Much of what we can do with lists, we can do with strings:

```
s = 'Hello, world!'
print(s)
>> Hello, world!
s_upper_list = [x.upper() for x in s]
print(s_upper_list)
>> ['H', 'E', 'L', 'L', 'O', ',', ' ', 'W', 'O', 'R', 'L', 'D', '!']
ss = ''.join(s_upper_list)
print(ss)
>> HELLO, WORLD!
```

`s.lower()` gives a string where all upper-case letters have been transformed into lower-case and `s.upper()` does the opposite.

`s.join(lst)` joins together the strings in `lst` inserting a `s` between each string.

Strings

Strings are sequences of characters. Much of what we can do with lists, we can do with strings:

```
s = 'Hello, world!'
print(s[-2:1:-2])
>> drw,l
print(type(s[2]))
>> <class 'str'>
```

If we use slicing and extracts an element of a string, we obtain a new string, containing a single character… not a value of a different data-type like in many other languages.

Strings

Strings are sequences of characters. Much of what we can do with lists, we can do with strings:

```
s1 = 'abc'
s2 = 'abd'
print(s1 < s2)
>> True
print(s2 < s1)
>> False
```

Comparisons of strings are done lexicographically (character by character from left to right)
[Also holds true for lists]

Strings are immutable

```
s = 'Hello, world!'
s[1:5]

>> 'ello'

>> s[1:5] = 'olle'

>> Traceback (most recent call last):
>>   File "<stdin>", line 1, in <module>
>>  TypeError: 'str' object does not support item assignment
```

You can not modify a string after it has been created, only create new strings with different content.

Strings are immutable

```
s = 'Hello, world!'
ss = s[0:1] + s[4:0:-1] + s[5:]

print(ss)
>> Holle, world!
```