Logic and Discrete Structures -LDS

Course 4 – Lists

Dr. Eng. Cătălin Iapă
e- mail: catalin.iapa@cs.upt.ro
Facebook : Catalin Iapa
cv: Catalin Iapa

Types of data

Python provides 4 types of primitive data :

- Integer
- Float
- String
- Boolean

The primitive data types in Python are immutable. This means that once they are created, their values cannot be changed.

If you assign a new value to a variable of a primitive data type, a new object is created with the updated value, rather than modifying the original object.

Predefined types for data collections

In Python, there are several predefined types for data collections.

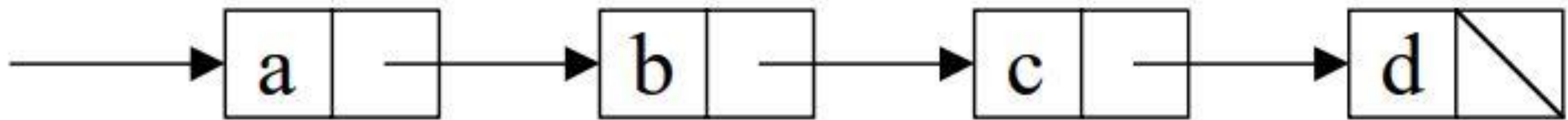
These types allow us to store and manipulate collections of data in a structured manner.

The four main predefined types for data collections in Python are:

- List
- Tuple
- Set
- Dictionary

Lists - representation

We can *represent* a list as a diagram of linked boxes:

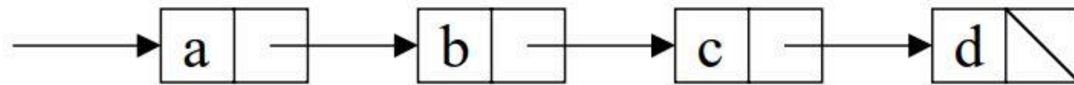


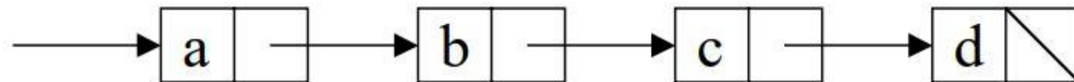
Above is a *list* with 4 elements:

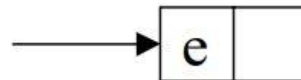
['a' , ' b', ' c', ' d']

Image : Robert M. Keller. Computer Science: Abstraction to Implementation, 2001

Adding an item to the list







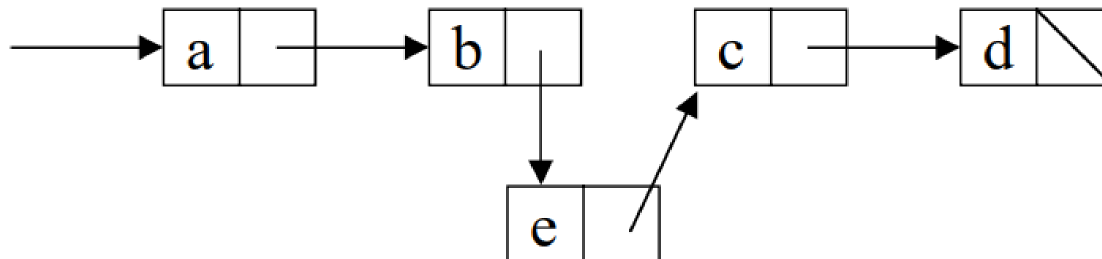


Image : Robert M. Keller. Computer Science: Abstraction to Implementation, 2001

Lists - representation

We can represent a list using item indexes.

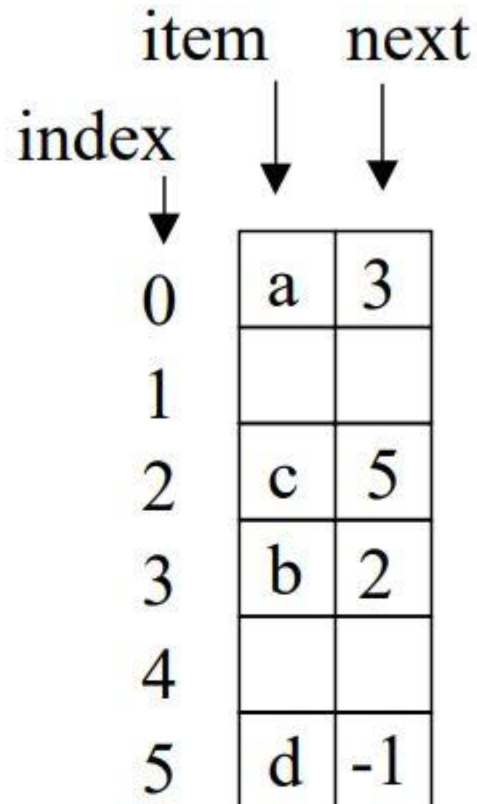


Image : Robert M. Keller. Computer Science: Abstraction to Implementation, 2001

Adding an item to the list

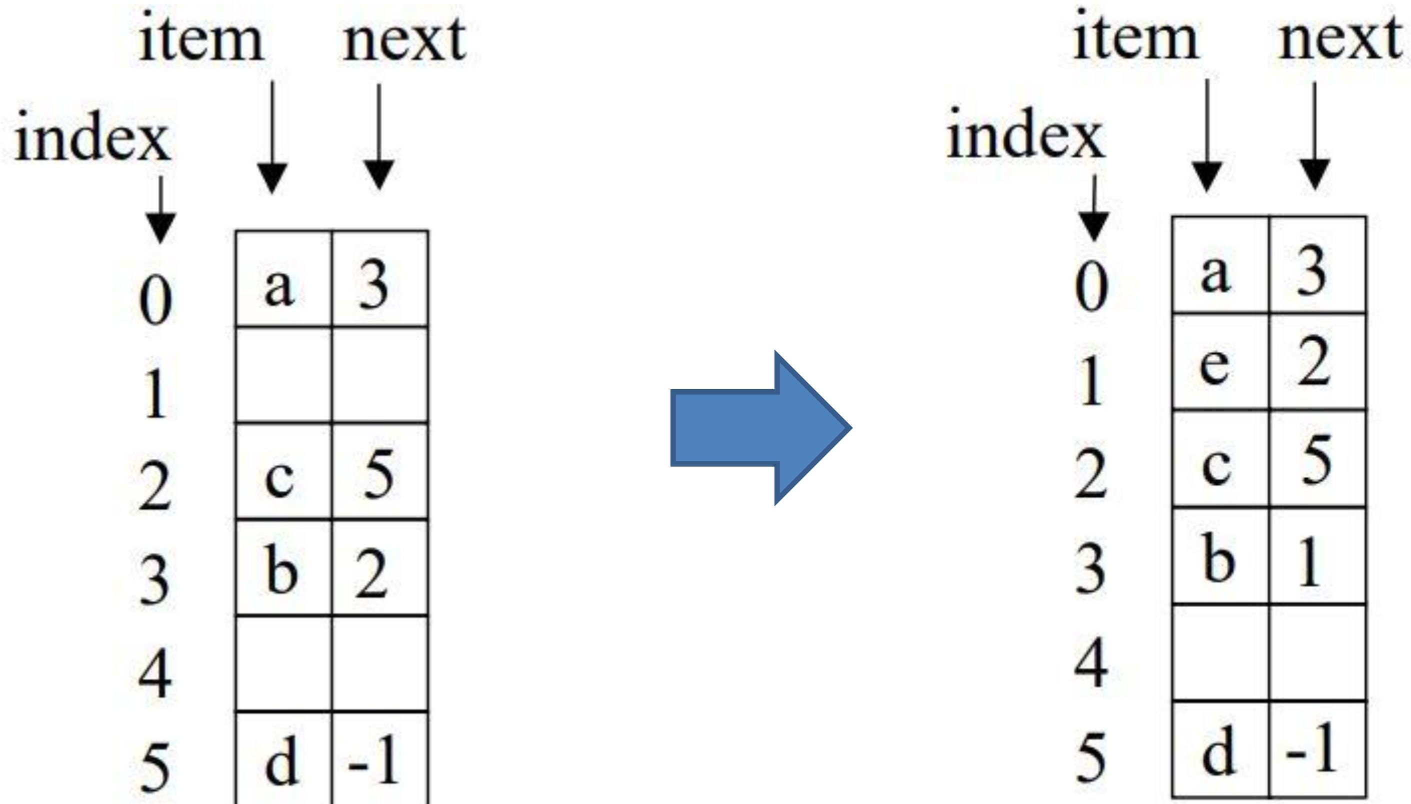


Image : Robert M. Keller. Computer Science: Abstraction to Implementation, 2001

Lists

Lists are one of the types that represent collections of items.
A list it is *a finite , ordered* sequence.

- The lists are *finite*, but they can be of any length
- *The order* elements matter: [1; 3; 2] ≠ [3; 1; 2]
- *Access* to the list elements it is *sequential* (direct access to the first only)
  - different from *vector/array* : *direct access* (with index) to any element

Lists

Two lists are equal if they have *the same elements* in the same *order* .

A list may have no elements, and we call it *an empty list* , or it may contain one or more elements.

We can decompose a list into:

- *The head of the list* - the first element
- *The tail of the list* - the other elements

List as a recursive type

Lists can be *defined recursively* :

a list is { an empty list
an *element* *followed by a* *list*
$(head)$ $(tail)$

Attention : the tail of the list is a list , NOT the last element .

The elements of a list

A list can contain as elements *any type of data* .

[4, 6, 8, 10, 12]

[“string”, “of”, “ characters ”]

A list can have elements including *other lists* .

[[4, 9], [1, 2, 3, 4], [19, 20]]

A list can contain elements of *different types* .

[8, 'a', [3, 6, 8], " word ", 9]

Lists in Python

A list is created using square brackets [ ] , like this :

list1 = ["Timisoara", "Arad", " Bucharest "]

list2 = [2022, 2023, 2024]

The length of a list

We can find *the length* of a list using the *len() function*

list1 = ["Timisoara", "Arad", " Bucharest "]

print (len (list1))

#3

How to create a list

We can *create a list* in several ways:

list2 = [2022, 2023, 2024]

A list can also be created using the constructor *list()* like this :

list2 = list ((2022 , 2023)) # similar [2022, 2023]

list3 = list ("sir") # similar to list3 = ['s', ' i ', 'r']

Accessing list elements

Elements can be accessed through *index* . Index of an element can be positive [0], [1], [2] etc. or negative [-1], [-2],[-3] etc.

list1 = [" Timisoara ", " Arad ", " Bucharest "]

```
print ( list1[0]) # Timisoara
print (list1[1]) # Arad
print (list1 [-1]) # Bucharest , the last element
print (list1 [-2]) # Arad
print (list1 [-3]) # Timisoara
```

Accessing list elements

We can access *multiple elements* in a list by specifying the index of the element at the beginning of the sequence and the element at the end of the sequence . The result is *a new list* containing the specified items.

list1 = ["Timisoara", "Arad", " Bucharest " , " Iasi "]

print (list1 [1:3])
 # will show ['Arad', ' Bucharest ']

 # the new list will start from element 1 to element 3. It will include the first element, but not the last

Accessing list elements

We can omit the start or end index.

If we omit the one at the beginning it will take over *starting with element 0* :

list1 = ["Timisoara", "Arad", " Bucharest " , " Iasi "]

print (list1 [:3])
 # it will display ['Timisoara ' , 'Arad ', ' Bucharest ']

If we omit the one at the end , it will take up *to the last element* :

print (list1 [2 :])
 # it will display [' Bucharest ' , ' Iasi ']

Accessing list elements

We can also use *negative indices* :

list1 = ["Timisoara", "Arad", " Bucharest " , " Iasi "]

print (list1 [-3 : -1])
 # will display ['Arad ', ' Bucharest ']

print (list1 [: -1])
 # will display ['Timisoara ' , 'Arad ', ' Bucharest ']

print (list1 [-2 :])
 # will display [' Bucharest ' , ' Iasi ']

Checking for the existence of an element

We can check if *an element is in a list* using the *in statement* :

```
list1 = ["Timisoara", "Arad", "Bucuresti", "Iasi"]
elem = "Arad"
if elem in list1:
        print(" the searched item is in the list ")
else:
        print(" the searched item is not in the list ")
```

Change an item in the list

We can *change* a list element by its index:

list1 = ["Timisoara", "Arad", "Bucharest" , "Iasi"]

list1 [1] = "Craiova"

The new list will be:

['Timisoara', 'Craiova', 'Bucharest' , 'Iasi']

Adding an item to the list

We can add a new element to the list without removing another element: we will use the *append() method* . The item will be added to *the end of the list* .

list1 = ["Timisoara", "Arad", " Bucharest "]

list1. append (" Resita ")

It is inserted in the last position. The new list:

[' Timisoara' , 'Arad', ' Bucharest ' , ' Resita ']

Adding an item to the list

We can add a new element to the list at a certain position, without removing another element, using the *insert() method* :

list1 = ["Timisoara", "Arad", " Bucharest "]

list1.insert (1, " Resita ")

It is inserted in position 1. New list will be:

[' Timisoara ', ' Resita ' , 'Arad', ' Bucharest ']

Adding an item to the list

Whether we use the *insert() method* or *the append() method* , the size of the list will increase by one element.

Adding items to another list

To add the elements of another list to the current list we will use the *extend() method* . The elements will be added *to the end of the current list* .

```
list1 = [ 3 , 4 , 5 ]
list2 = [ 101 , 102 _ 110 ]
list1.extend ( list2 )
print ( list1 )
```

 # will display [3 , 4 , 5, 101 , 102 , 110]

Removing items from the list

To remove a certain element from the list we will use the *remove() method* .

```
list1 = [ 1, 2, 3 , 4 , 5 ]
list1.remove ( 2 )
print ( list1 )

# will display [ 1, 3 , 4 , 5 ]
```

Removing items from the list

remove() method will only remove *the first occurrence* of the element in the list.

list1 = [1, 2, 3 , 4 , 5, 2]
list1.remove (2)
print (list1)

 # will display [1, 3 , 4 , 5, 2]

Removing items from the list

We can *remove* an element by specifying its *index* with the *pop() method*

list1 = [1, 2, 3 , 4 , 5, 2]
list1.pop (2)
print (list1)

 # will display [1, 2 , 4 , 5 , 2]

Removing items from the list

If *we do not specify the index* within the pop() method, *the last element* of the list will be removed.

Removing items from the list

If *we do not specify the index* within the pop() method, *the last element* of the list will be removed.

list1 = [1 , 2 , 3 , 4 , 5]
list1.pop ()
print (list1)

 # will display [1, 2 , 3, 4]

Removing items from the list

To delete an item from the list we can use *del*

```
list1 = [ 1 , 2 , 3 , 4 , 5 ]
del list1 [0]
print ( list1 )

# will display [ 2 , 3, 4 , 5]
```

We can also delete the entire list:

```
del list
```

Removing items from the list

If we want to remove all elements from a list we use the *clear() method*

```
list1 = [ 1, 2, 3 , 4 , 5 ]
list1. clear ()
print ( list1 )

# will display the empty list: []
```

Sorting a list

To *sort* the elements of a list we will use the *sort() method* . This method implicitly sorts the items in the list in *ascending or lexicographical order*.

numbers = [1, 3, 2 , 6, 5, 4]

words = [' one ', ' two ' , ' three ']
numbers. sort ()

words. sort ()

print (numbers , words)

will display :

[1, 2, 3, 4, 5, 6]

[' two ', ' three ', ' one ']

Sorting a list

To sort the elements of a list *in descending order* we will use the sort() method with the argument *reverse = True* .

numbers = [1, 3, 2 , 6, 5, 4]
words = [' one ', ' two ' , ' three ']
numbers. sort (reverse = True)
words. sort (reverse = True)
print (numbers , words)

will display :
[6, 5, 4, 3, 2, 1]
[' one ', ' three ' , ' two ']

Sorting a list

We can use *specific criteria to sort* a list using the *key = function argument* . It will first apply the function to each element of the list and then sort by the result of the function.

```
def function ( x ):
        return abs( x - 1 0)

list1 = [ 1 , 2 , 10 , 11 , 2 9 ]
list1 . sort ( key = function )
print ( list1 )
```

will display :

[10, 11, 2, 1, 29]

Reverse the order of a list

To *reverse the order* of the elements of a list we use the *reverse() method*

numbers = [1, 2, 3, 4, 5, 6]

numbers. reverse ()
print (numbers)

will display :
[6, 5, 4, 3, 2, 1]

Copying a list

To copy a list to another list we will use the *copy() method* or the *list() constructor*

list1 = [1, 2, 3, 4, 5, 6]

list2 = list1. copy ()
list3 = list (list1)

If we use the assignment operator =, it will not copy the contents of one list to another list. It will just be a *reference to the first list*, and any changes to the first list will be reflected if we use the new object.

Concatenation of two lists

Using *the + operator* for 2 lists we will concatenate their contents *into a new list* .

```
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]

list3 = list1 + list2
print (list3)

# will display [ 'a', 'b', 'c', 1, 2, 3]
```

Functions for lists: map()

We can apply a function to each element of a list with the *map() function* .

map() function has 2 arguments, the first is a *function* and the second is *the list* to which the function applies.

new_list = map (function, list)

Functions for lists: map()

Example:

```
def square ( x ):
    return x*x

num = [1 , 2, 3, 4]
list1 = map ( square , num )
print ( list ( list1 ))
```

will display:

[1, 4 , 9 , 16]

Functions for lists: map()

We can also use *the anonymous function* in *map()* .

num = [1, 2, 3, 4]
*list1 = map (lambda x: x * x , num)*

We can also apply map() to *several lists* in parallel:

num1 = [1, 2, 3]
num2 = [4, 5, 6]
result = map (lambda x , y: x + y, num1 , num2)

https://www.geeksforgeeks.org/python-map-function/

Functions for lists: reduce()

reduce() function sequentially applies a given function to the elements of a list.

It has 2 arguments and returns the result of the sequential application of the function

result = reduce (function, list)

The function is defined in the *functools module* .

Functions for lists: reduce()

the reduce() function *works* :

- In the first step, the function is applied to *the first 2 elements in the list* and the result is retained
- After which the function is applied *with the result* obtained in the previous step and *the next element* in the list and the result is retained
- The previous step *is repeated* until all items in the list are covered and *the final result is returned*

Functions for lists: reduce()

Example of use:

```
import functools

list1 = [1, 3, 5, 6, 2]

print ( " The sum of the elements of the list is : " )
print ( functools.reduce ( lambda a, b: a+b , list1 ))

print (" The maximum element is : ")
print ( functools . reduce ( lambda a, b: a if a > b else b, list1 ))
```

https://www.geeksforgeeks.org/reduce-in-python _

Functions for lists: reduce()

Example of use:

```
import functools

list1 = [1, 3, 5, 6, 2]

import functools

list1 = [1, 3, 5, 6, 2]

print ( functools . reduce ( lambda a, b: a if a < b else b, list1 ))
```

https://www.geeksforgeeks.org/reduce-in-python _

Functions for lists: filter()

The function *filter(function, list)* tests each element of the list with the given function and returns only the elements that satisfy the condition in the function.

The function received as a parameter must return *True* or *False*

The result will only contain the elements for which the function *returns True*

Functions for lists: filter()

Example of use:

```
def function ( letter ) :
        vowels = ['a', 'e', ' i ', 'o', 'u']
        if ( letter in vowels ):
                return True
        else :
                return False

list1 = ['g', 'e', 'e', 'j', 'k', 's', 'p', 'r ']
list2 = filter ( function , list1 )
print ( list (list2) )
```

https://www.geeksforgeeks.org/filter-in-python _

Functions for lists: filter()

Example of use:

```
numbers = [0, 1, 2, 3, 5, 8, 13]

odd = filter ( lambda x: x % 2 != 0, numbers )
print ( list ( odd ))

even = filter ( lambda x: x % 2 == 0, numbers )
print ( list ( even ))
```

It will display:
[1, 3, 5, 13]
[0, 2, 8]

https://www.geeksforgeeks.org/filter-in-python _

Memory address of the elements

To get the memory address for an item in a list we use the id() function.

Example:

list1 = [0, 1, 2, 3, 5, 8, 13]

print (id (list1[0]))

print (id (list1[1]))

will display:

2163888750800

2163888750832

index	item	next
0	a	3
1		
2	c	5
3	b	2
4		
5	d	-1

Memory address of the elements

As a rule, the memory address of a byte is expressed in base 16. To display the address in base 16 we can use the *hex() function*

Example:

list1 = [0, 1, 2, 3, 5, 8, 13]

print (hex (id (list1 [0])))

print (hex (id (list1 [1])))

will display:

0x227882c00d0

0x227882c00f0

index	item	next
0	a	3
1		
2	c	5
3	b	2
4		
5	d	-1

Head and tail of the list (Head , Tail)

If we have a list and we want to extract *the head of the list and* its tail we can write:

list1 = [0, 1, 2, 3, 4, 5]

head , tail = list1[0], list1[1:]

 # head = 0

 # tail= [1, 2, 3, 4, 5]

Recursive functions – Create list

```
def create_list_recurse(start, end):
    if start > end:
        return []
    return [start] + create_list_recurse(start + 1, end)

create_list_recurse(0, 9)
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

create_list_recurse(3, 1) #edge case returns empty
# []
```

https://stackoverflow.com/questions/62854078/how-to-create-a-recursive-function-to-create-a-list-of-values

Recursive functions – Number of elements

Recursion:

```
def length ( list1 ):
    if list1 == []:
        return 0
    return 1 + length(list1[1:] )

print(length ( [0, 1, 2, 3, 4, 5]))
# 6
```

Tail recursion:

```
def length2(list1, no=0):
    if (list1 == []):
        return no
    return length2(list1[1:], 1 + no)

print(length2 ([0, 1, 2, 3, 4, 5]))
# 6
```

Recursive Functions – Contains the element

The recursive function that tells us if an element is in a list or not:

```
def contains (x, list1 ):
   if ( list1 == []):
      return False
   return x == list1[0] or contains (x , list1[1 :])

print ( contains (4,[1,2,3 ]))
#False
```

To know

Lists are the simplest type of collection

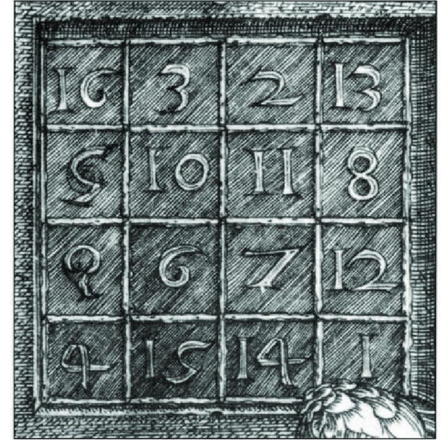
- exist in many programming languages

Working with standard list traversal functions

- how to simply write "*do this operation on the whole list*"

Operations that have functions as parameters

- allow us to specify the desired processing



Thank you!

Bibliography

- The content of the course is mainly based on the materials of the past years from the LSD course, taught by Prof. Dr. Marius Minea et al. Dr. Eng. Casandra Holotescu (http://staff.cs.upt.ro/~marius/curs/lsd/index.html)