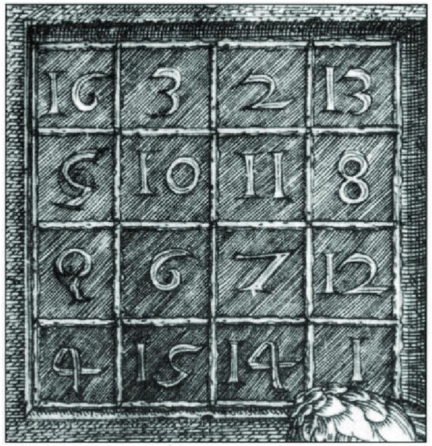
Logic and Discrete Structures - LDS



Course 5 – Relations. Dictionaries

dr. ing. Cătălin Iapă

catalin.iapa@cs.upt.ro

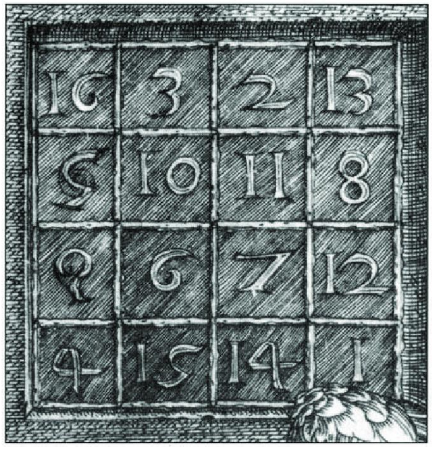
What have we learned so far?

Functions

Recursive Functions

Lists

Sets



Relations - theoretical aspects

Binary relations

Composition of relations

Dictionaries in PYTHON

Relations implemented with Dictionaries

Exercises with dictionaries

Relation - in the real world and computer

A (mathematical) relation models the connection between two entities (possibly of different types).

Examples:

Subject-object relations: a man read a book

Human relations: child , parent , friend

Quantitative relations : equal, lesser

Relation - in the real world and computer

Translated into computer science:

Social networks : "friend", "follow", "in circles", etc.

A relation between elements of the same set defines a graph

(elements are nodes, the relation is represented by edges)

⇒ relations are a key notion in graph theory

Relations - sets of pairs

A binary relation R between two sets A and B is the set of pairs: a subset of the Cartesian product

$A \times B$: $R \subseteq A \times B$

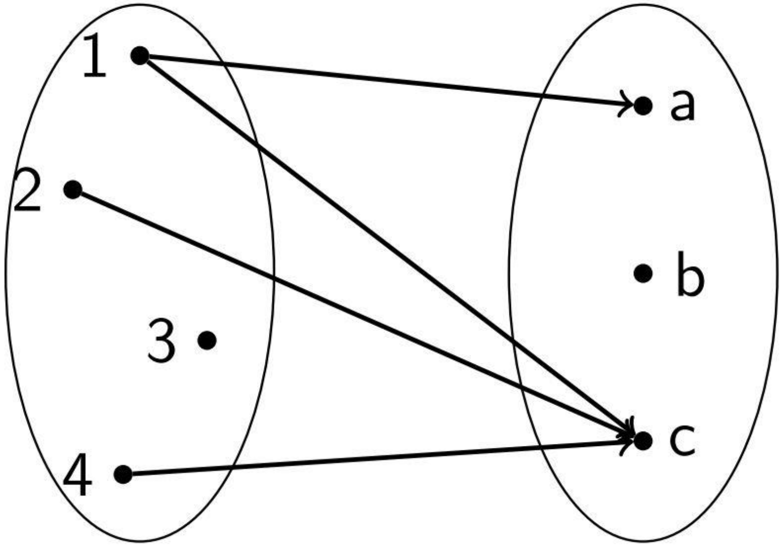
Denoted $(x, y) \in R$ or xRy or $R(x, y)$

when x is in relation to y

$A = \{1, 2, 3, 4\}$,

$B = \{a, b, c\}$

$R = \{(1, a), (1, c), (2, c), (4, c)\}$



Relations - sets of pairs

A relation is a more general notion than a function:

- a function associates to each x ∈ A a single y ∈ B

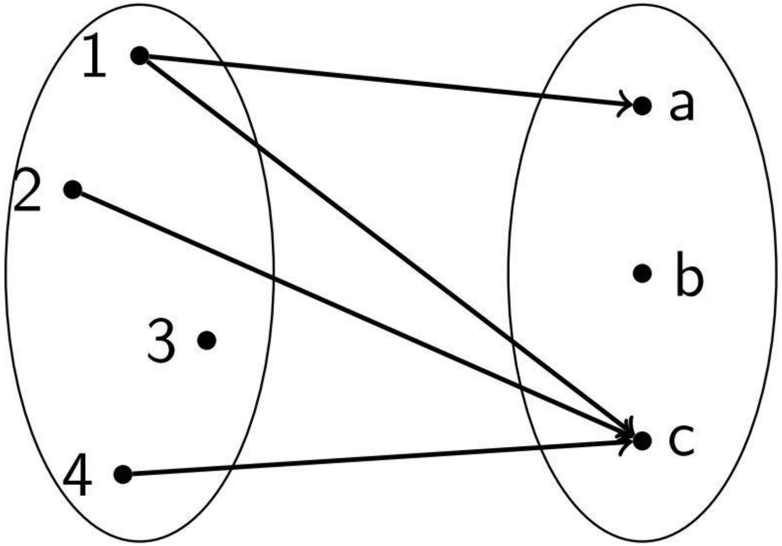
In a relation we can have

(see figure):

1: has several elements associated: a, c

2: has only one element associated: c

3: has no associated element from B



Relations - general aspects

In general, a relation is not a symmetric notion: the Cartesian product/pair are ordered notions,

$$(x, y) \neq (y, x)$$

There are, of course, symmetric relations (in the real world and in mathematics)

Generalized, we can have an n-ary relation that is a n-tuples set (from the Cartesian product of n sets).

Example:

$R \subseteq Z \times Z \times Z$

R(x, y, m) if m is a common multiple of x and y:

R(2, 9, 18), R(6, 9, 18), R(2, 9, 36), etc.

Representation of a relation

We can represent a relationship:

1. Explicitly, by the set of pairs (if finite)

$$R \subseteq \{1, 2, 3, 4\} \times \{a, b, c\}$$

$$R = \{(1, a), (1, c), (2, c), (4, c)\}$$

2. By a rule connecting the elements:

$$R = \{(x, x^2 + 1) \mid x \in Z\}$$

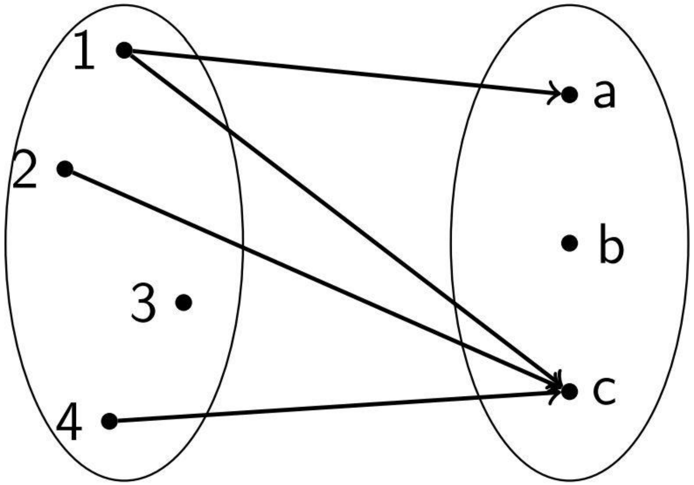
Representation of a relation

3. As a Boolean/binary matrix, if A, B finite,
rows indexed by A, and columns by B

m_{xy} = *1 if (x, y) ∈ R,*

m_{xy} = *0 if (x, y) ∉ R*

In practice we can use this type of representation if A and B are not very large.



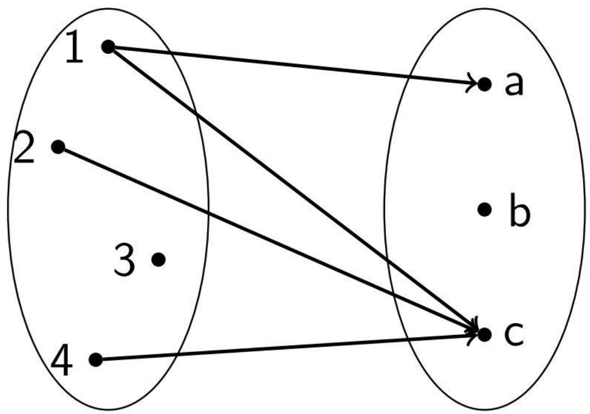
	a	b	c
1	1	0	1
2	0	0	1
3	0	0	0
4	0	0	1

Relation seen as a function

A relation $R \subseteq A \times B$ can be seen as a function f_R from A to the set of parts of B

$$f_R(x) = \{y \in B \mid (x, y) \in R\}$$

Associate each x with the set of the elements of B to which x is related (possibly empty): $f_R(1) = \{a, c\}$, $f_R(3) = \emptyset$



A vector of bits/booleans can represent a set :

a	b	c
1	0	1

represents {a, c} (by characteristic function)

Number of relations between two sets

Between A and B (finite) there are $2^{|A|\cdot|B|}$ relations $R \subseteq A \times B$

It follows directly from the definition: a relation is a subset $R \subseteq A \times B$. So, $R \in P(A \times B)$.

But $|P(A \times B)| = 2^{|A \times B|} = 2^{|A|\cdot|B|}$.

Or, using the representation as a matrix, which has "|A|*|B|" elements. each: chosen independently in 2 ways: 0 or 1, so $2^{|A|\cdot|B|}$ choices.

Partial functions

A partial function f : A → B is a particular case of relation: associates a single element of B (as the function) but not necessarily every element of A (as the function is bound to)

Partial functions are useful:

- when the exact domain of the function is not known(functions that are not necessarily computable at any point).
- when the domain of definition of the function is very large or unlimited, but we represent the function explicitly only for the values of interest

Example: population of a locality

- we may not know the population for all localities
- if the argument is a string, not every string is a locality name

Relations - theoretical aspects

Binary relations

Composition of relations

Dictionaries in PYTHON

Relations implemented with Dictionaries

Exercises with dictionaries

Binary relations on a set

The following properties are defined for binary relations on a (same) set X: $R \subseteq X \times X$

- *reflexive: for any $x \in X$ we have $(x, x) \in R$*
- *irreflexive: for any $x \in X$ we have $(x, x) \notin R$*
- *symmetric: for any $x, y \in X$, if $(x, y) \in R$ then also $(y, x) \in R$*
- *antisymmetric: for any $x, y \in X$, if $(x, y) \in R$ and $(y, x) \in R$, then $x = y$*
- *transitive: for any $x, y, z \in X$, if $(x, y) \in R$ and $(y, z) \in R$, then $(x, z) \in R$*

Properties of binary relations

What properties do the following relations have?

Property / Relation	reflexive	symmetric	antisymmetric	transitive
$x \equiv y \pmod{n}$	Yes	Yes	NU	Yes
$x \mid y$	Yes	No	Yes	Yes
$x \leq y$	Yes	No	Yes	Yes

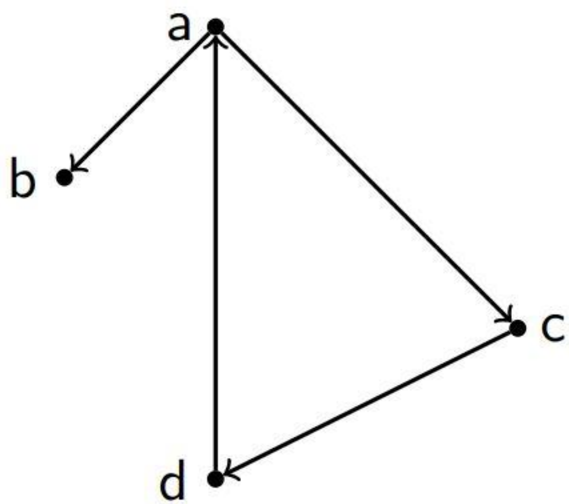
Binary relations and graphs

A binary relation on a set X can be represented as a graph with X as a set of nodes:

Directed graph:

random relation

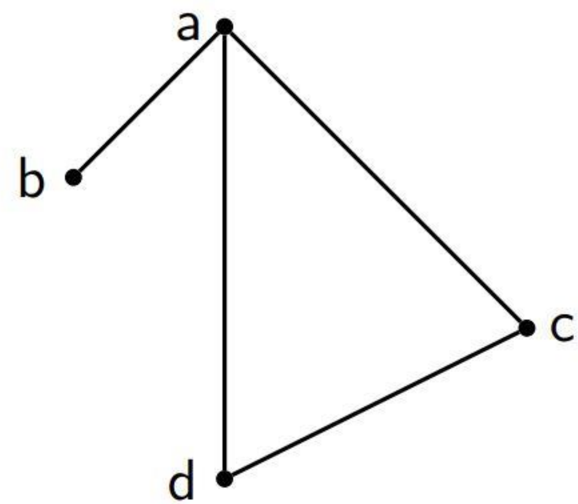
$R = \{(a,b), (a,c), (c,d), (d,a)\}$



Undirected graph:

Symmetric relation

$R = \{(a, b), (a, c), (a, d), (b, a), (c, a), (c, d), (d, a), (d, c)\}$



Equivalence relations

A relation is an equivalence relation if it is reflexive, symmetric and transitive

The relation of equality is (obviously) an equivalence relation.

The congruence relation modulo a number (mod n):

a ≡ b (mod n) if n | a - b (divide the difference)

The equivalence class of x is the set of elements related to x:

$\{y \mid (y, x) \in R\}$ denoted $\hat{x}$ or $[x]$

Strict order relations

A relation $\prec$ is a strict order if it is irreflexive and transitive

- there is no x with $x \prec x$
- if $x \prec y$ and $y \prec z$ then $x \prec z$

Examples:

- relations < and > between numbers
- - the "descendant" relation between persons

Total order relations

A relation ≤ is a total order if it is:

- reflexive,
- antisymmetric (if x ≤ y and y ≤ x then x = y),
- transitive, and in addition any two elements are comparable, i.e. for any x , y we have x ≤ y or y ≤ x

Examples: relations ≤ and ≥ between numbers (integers, reals, etc.)

Partial order relations

In practice, relations of order often arise that are not total:

- ranking within groups, but not between different groups
- We know the order in which messages arrive, but not the order in which they are sent
- in the expression f (x) + g (x), f and g are called before addition, but we do not know whether f or g is evaluated first

A relation is a partial (non-strict) order if it is: reflexive, antisymmetric and transitive

Examples:

The divisibility relation between integers

Inclusion relation $\subseteq$ on the set of parts

Partial order relations

Any total order is also a partial order (but not reciprocally).

Any partial order induces a strict order, and reciprocally:

We define: $a \prec b$ *if* $a \leq b$ *and* $a \neq b$

Conversely, we define $a \leq b$ *if* $a \prec b$ *or* $a = b$

Properties of binary relations

Property Relation	reflexive	symmetric	antisymm.	transitive	
$x \equiv y \pmod{n}$	Yes	Yes	No	Yes	Equivalence relation
x | y	Yes	No	Yes	Yes	Partial order relations
x ≤ y	Yes	No	Yes	Yes	

Relations - theoretical aspects

Binary relations

Composition of relations

Dictionaries in PYTHON

Relations implemented with Dictionaries

Exercises with dictionaries

The inverse of a relation

The inverse of a relation $R \subseteq A \times B$ *is the relation*

$$R^{-1} \subseteq B \times A,$$

with $(y, x) \in R^{-1}$ if and only if $(x, y) \in R$

$$R^{-1} = \{(y, x) \mid (x, y) \in R\}$$

Composition of relations

Two relations $R_1 \subseteq A \times B$ and $R_2 \subseteq B \times C$.

Composition $R_2 \circ R_1 \subseteq A \times C$ is the relation

$R_2 \circ R_1 = \{(x, z) \mid \text{exist } y \in B \mid (x, y) \in R_1 \text{ și } (y, z) \in R_2\}$

As with functions, we write R2 ∘ R1 and see that for x ∈ A we first find y ∈ B and then z ∈ C .

Composition of relations

We can see that $(R \circ S)^{-1} = S^{-1} \circ R^{-1}$

For an equivalence relation R, $R = R^{-1}$

R is transitive if and only if $R \circ R \subseteq R$

For a binary relation $R \subseteq A \times A$, denote
$R^2 = R \circ R$, etc.

Relations - theoretical aspects

Binary relations

Composition of relations

Dictionaries in PYTHON

Relations implemented with Dictionaries

Exercises with dictionaries

Dictionaries in PYTHON

The dictionary is a collection:

- *ordered (as of Python version 3.7),*
- *changeable after creation and*
- *does not allow duplicates.*

Dictionaries are used to store data in key:value pairs.

Dictionaries in PYTHON

Dictionaries are written between two curly braces {} and have comma-separated key:value pairs as elements.

```
dict1 = {
  "name": "Alin", “year": 1,
  "faculty": "Automatica si Calculatoare"
}
print(dict1)
```

{‘name': 'Alin', ‘yesr': 1, ‘faculty': 'Automatica si Calculatoare'}

Dictionaries in PYTHON

Values in the key-value pair can be any data type and can be repeated.

Keys in the key-value pair can only be data that cannot be changed after their creation (immutable) and cannot be repeated.

```
dict1 = {}
dict2 = {1: “one", 2: “two"}
dict3 = {
        “name": "Ana",
        “children": ["Andrei", "Maria"]
}
```

Dictionaries in PYTHON

We can also create dictionaries with the constructor dict()

```
dict1 = dict()
dict2 = dict({1: “one", 2: “two"})
dict3 = dict(((10, “ten"), (100, “one hunderd")))

# {}
# {1: ‘one', 2: ‘two'}
# {10: ‘ten', 100: ‘one hunderd'}
```

Accessing dictionary elements

If in lists we use indexes to access elements, in dictionaries we use keys. To access an element we use square brackets [] or the get() method.

```
dict1 ={
    "name": "Alin", “year": 1,
    "faculty": "Automatica si Calculatoare"
}
print(dict1[“year"])                    # 1
print(dict1.get("name"))                # Alin
```

Accessing dictionary elements

To access the elements we can use the methods keys(), values() and items() as follows:

```
dict1 ={"name": "Alin", “year": 1, "faculty": "AC"}
print(dict1.keys())
print(dict1values())
print(dict1.items())

# dict_keys(['name', ‘year', 'faculty'])
# dict_values(['Alin', 1, 'AC'])
# dict_items([('name', 'Alin'), (‘year', 1), ('faculty', 'AC')])
```

Adding elements to the dictionary

Dictionaries can be modified after they have been created: we can add new elements or modify the value of an existing key.

dict1 ={"name": "Alin", “year": 1, "faculty": “AC"}

dict1["name"] = "Marius"
dict1[“age"] = 20

print(dict1)
 # {'name': 'Marius', ‘year': 1, 'faculty': AC', ‘age': 20}

Adding elements to the dictionary

We can add new elements or modify existing elements using the update() method

dict1 ={"name": "Alin", “year": 1, "faculty": "AC"}

dict1.update({"name":"Marian"})

dict1.update({"surname": "Popescu", “grade": 10})

print(dict1)

#{'name': 'Marian', ‘year': 1, 'faculty': 'AC', 'surname': 'Popescu', ‘grade': 10}

Deleting elements from the dictionary

To delete elements from the dictionary we can use the methods:

- pop() - deletes the element specified as a parameter,
- popitem() - delete a random element from the
- clear() - clear all items in the dictionary

```
dict1 = {"name": "Alin", “age": 20, “year": 1, "faculty": "AC"}
dict1.pop("faculty")
print(dict1)              # {'name': 'Alin', ‘age': 20, ‘year': 1}
dict1.popitem()
print(dict1)              # {'name': 'Alin', ‘age': 20}
dict1.clear()
print(dict1)              # {}
```

Deleting elements from the dictionary

We can delete individual elements or the entire dictionary with del

dict1 = {"name": "Alin", “age": 20, “year": 1, "faculty": "AC"}

del dict1['name']

print(dict1)   # {‘age': 20, ‘year': 1, 'faculty': 'AC'}

del dict1

print(dict1)   # NameError: name 'dictionar' is not defined.

Checking the existence of an element

To check if a key exists in the dictionary we use in.
We cannot search by value but only by key.

```
double = {1: 2, 2: 4, 3: 6, 4: 8, 5: 10}

x = 2
if(x in double):
    print(" the key is in the dictionary")
else:
    print(" the key is not in the dictionary")
```

Nested dictionary

We can have a dictionary as an element of another dictionary (nested dictionary)

```
dict1 = {
    "dict2": {1: 1, 2: 4, 3: 9},
    "dict3": {1: “one", 2: “two"}
}

print(dict1["dict2"][3])
print(dict1["dict3"][2])

# 9
# two
```

Relations - theoretical aspects

Binary relations

Composition of relations

Dictionaries in PYTHON

Relations implemented with Dictionaries

Exercises with dictionaries

Relations using dictionaries

We have seen that a relation $R \subseteq A \times B$ can be seen as a function f_R from A to the set of parts of B

$$f_R(x) = \{y \in B \mid (x, y) \in R\}$$

Associate each x with the set of elements of B to which x is related (possibly empty):

$f_R(1) = \{a, c\}$, $f_R(3) = \emptyset$

The dictionary will then be from A to subsets of elements in B.

Relations using dictionaries

relation = {

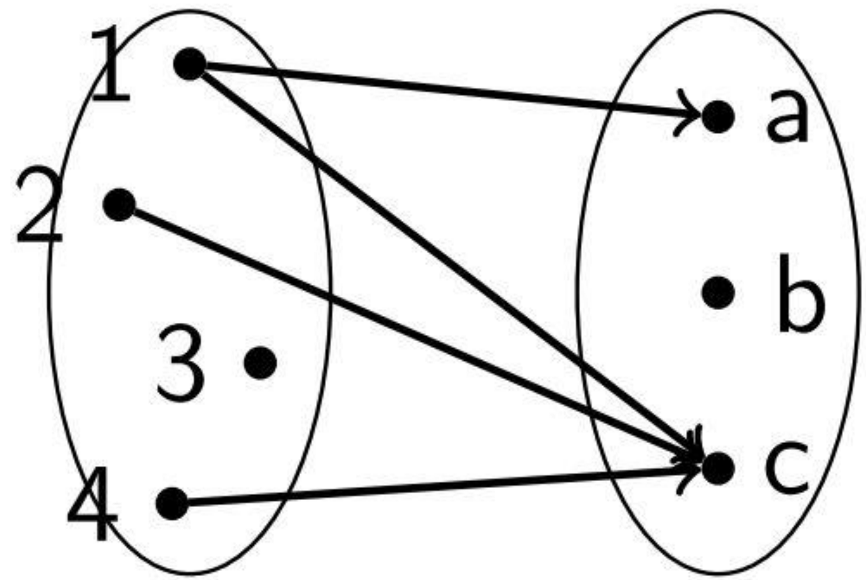
1: {"a", "c"},

2: {"c"},

3: set()

4: {"c"}

}



#{1: {'a', 'c'}, 2: {'c'}, 3: set(), 4: {'c'}}

Relations - theoretical aspects

Binary relations

Composition of relations

Dictionaries in PYTHON

Relations implemented with Dictionaries

Exercises with dictionaries

Exercises with dictionaries

1. Write a function that takes an association list with pairs of type (string, integer) and creates a dictionary where each string is associated with the sum of all values it is associated with in the list.

Exemple:

Input: [("Ana",7), ("Alin",3), ("Ana",9)]

Output: {'Ana': 16, 'Alin': 3}

Exercises with dictionaries

```
def transform(lista, dictionar = {}):
    if (lista == []):
        return dictionar
    if(lista[0][0] in dictionar):
        dictionar[lista[0][0]] = lista[0][1] + dictionar[lista[0][0]]
    else:
        dictionar[lista[0][0]] = lista[0][1]
    return transform(lista[1:],dictionar)

l = [("Ana",7), ("Alin",3), ("Ana",9)]

print(transform(l))
```

Exercises with dictionaries

2. Dictionary traversal using the reduce() function:

```
elev_nota = {
    'Alex': 10,
    'Mihai': 9,
    'Ioana': 10
}

print(elev_nota.items())
# dict_items([('Alex', 10), ('Mihai', 9), ('Ioana', 10)])
```

Exercises with dictionaries

Dictionary traversal using the reduce() function:

```
def functie_suma(suma, elev):
        nume, nota = elev
        return suma + nota

def medie_elevi(dictionar):
        suma_note = functools.reduce(functie_suma, dictionar.items(), 0)
        return suma_note / len(dictionar)

print(medie_elevi(elev_nota))
```

Exercises with dictionaries

3. Recursive dictionary traversal.

For recursive dictionary traversal, we convert the dictionary received as a parameter to 'dict_items', then convert 'dict_items' to a list that we will recursively traverse.

```
elev_nota = {
    'Alex': 10,
    'Mihai': 9,
    'Ioana': 10
}
```

Exercises with dictionaries

```
def suma_recursiva(dict_list):
    if len(dict_list) > 0:
        nume, nota = dict_list[0]
        return nota + suma_recursiva(dict_list[1:])
    else:
        return 0

def medie_elevi_recursiva(dictionar):
    suma_note = suma_recursiva(list(dictionar.items()))
    return suma_note/len(dictionar)

print(medie_elevi_recursiva(elev_nota))
```

Thank you!

Bibliography

- The content of the course is mainly based on the material from the LSD course taught by Prof. Dr. Eng. Marius Minea and S.l. Dr. Eng. Casandra Holotescu (http://staff.cs.upt.ro/~marius/curs/lsd/index.html)