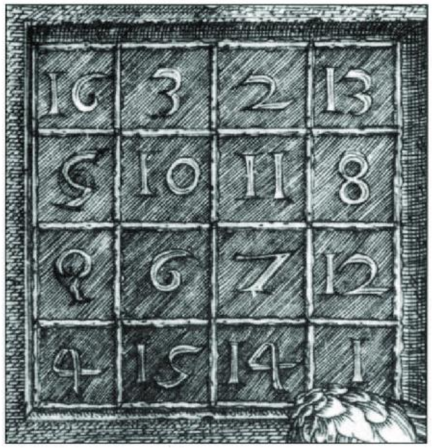
Welcome to the **Logic and Discrete Structures - LSD course**



Course 1

Dr. Eng. Cătălin Iapă

catalin.iapa@cs.upt.ro

Let's get to know:

Course:

Cătălin Iăpa

Laboratories:

Andrei Deac

Brumar Raul

Let's get to know each other

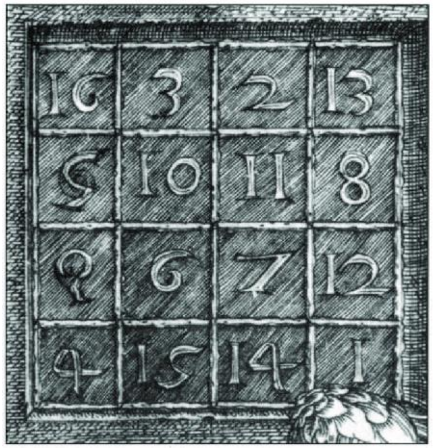
- Moldova-Noua, Caraș-Severin
- Grigore Moisil High School Timisoara
- Faculty of Automation and Computers, UPT
  - Computers and information technology
- PHD
  - Identifying the user of a computer based on the way he types on the keyboard
- AC League, Youth House, Timisoara City Hall, Ministry of Development, Authority for Digitization of Romania
- Computer Programming, Programming Techniques, Software Project Management, Logic and discrete structures

Course administrative details

- Semester 1: 14 weeks
  - 2 hours of class / week
  - laboratory hours / week

- You will receive 2 grades , their average is the final grade at LSD
  - 1 mark in the exam, in the session, after the 14 weeks
  - 1 mark for the laboratory activity

- To pass the subject, you need to get at least grade 5 , both in the exam and in the laboratory
- The exam can be taken 3 times



What we do on LSD

Demonstrations

Sets

Functions

Properties of Functions

Functions in Programming

About LSD

- Logic and Discrete Structures
- What are we going to do?
  – LOGIC,
  – MATH and *What Kind of Math?*
  – PROGRAMMING *What kind of Programming?*
- What do you need before starting this course?
  – Fundamentals of mathematics
  – Curiosity

About LSD

MATHEMATICAL LOGIC

- how do we express sentences precisely
  - for rigorous definitions , software specifications
- how do we prove statements
  - to show that an algorithm is correct
- how do we process logical formulas
  - to find solutions to problems

About LSD

DISCRETE MATHEMATICS

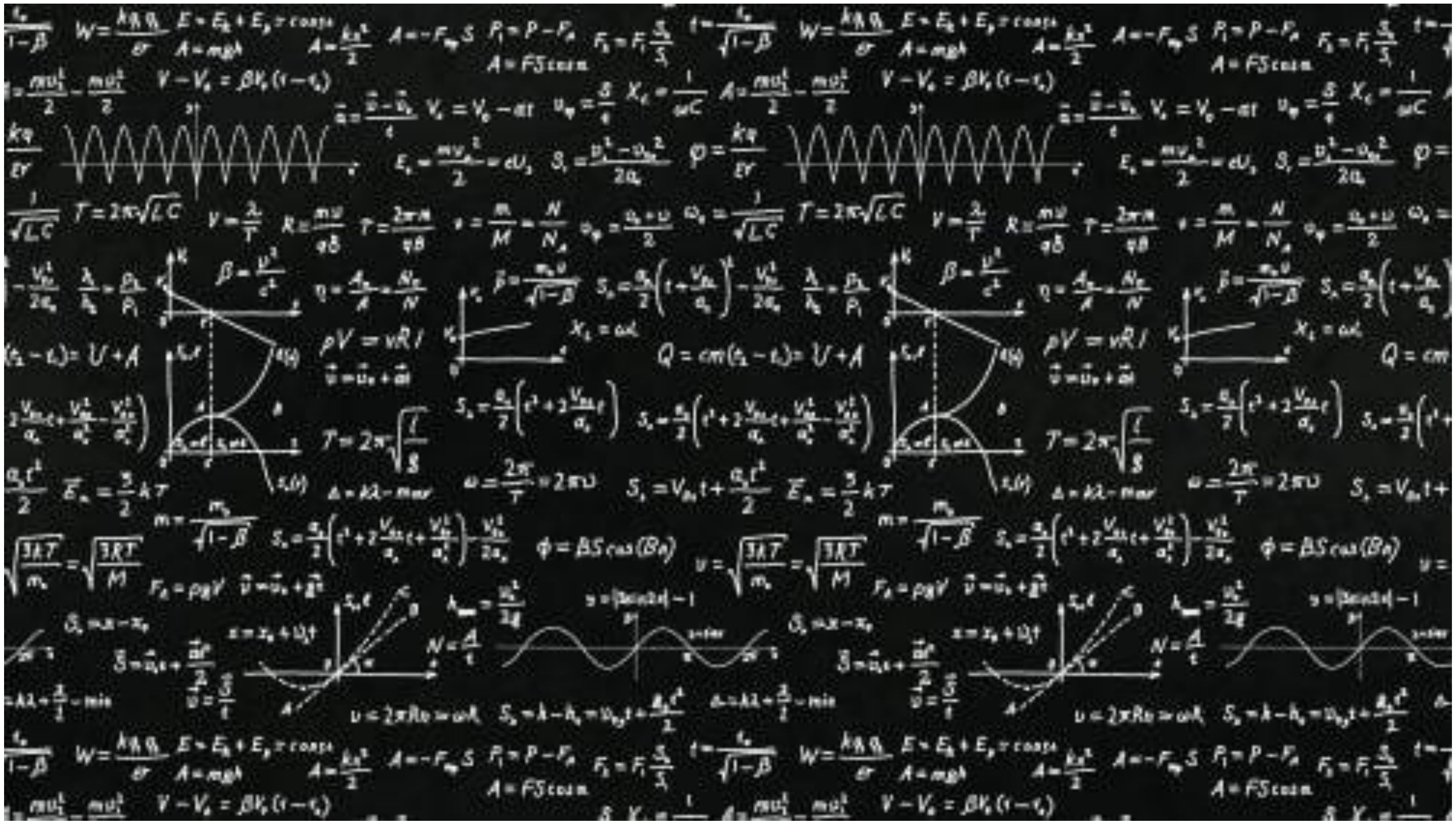


Image source: https://engineering.jhu.edu/ams/wp-content/uploads/2021/06/hero-image-research-500x282.jpeg

About LSD

DISCRETE MATHEMATICS

It is the basic language of computer science

- *algorithms*
- *bioinformatics*
- *computer graphics*
- *data science*
- *machine learning*
- *software engineering etc.*

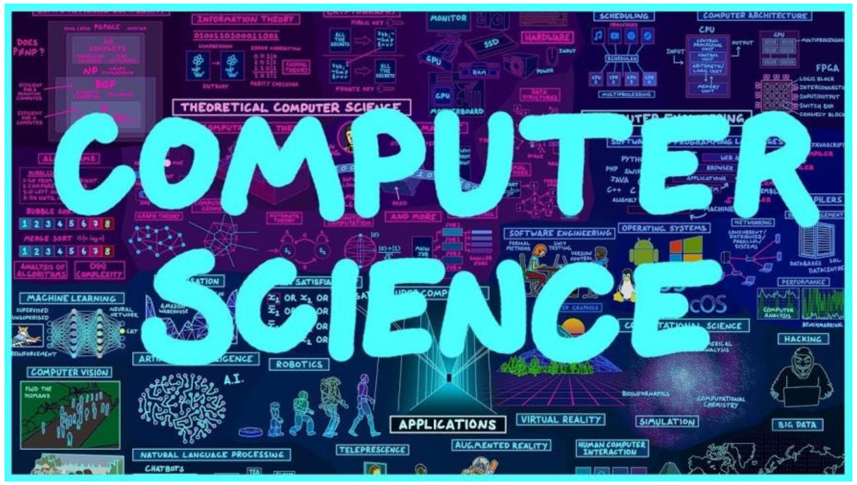


Image source: https://img.youtube.com/vi/Jv34MWng28o/maxresdefault.jpg

About LSD

DISCRETE MATHEMATICS

- What are we studying?

we study notions/objects that take distinct, discrete values

- integers, logical values, relations, trees, graphs, etc.

- What are we not studying?

we do not study the continuous field

- real numbers, infinitesimals, limits, differential equations see: mathematical analysis

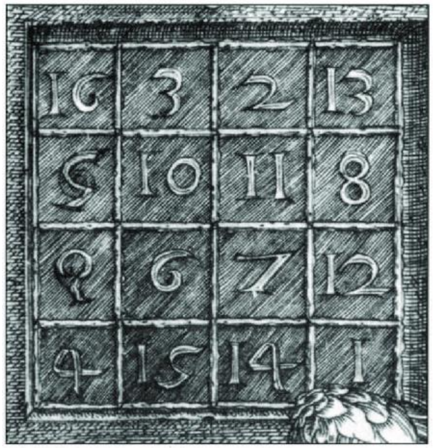
About LSD

PROGRAMMING in PYTHON

PYTHON programming language

High level language

- It's super easy to get started with Python (even if you've never programmed before)
- The syntax is reader-friendly (and close to natural language)
- The code is compact
- The Python standard library provides a wide range of facilities and many external libraries are also available
- It is one of the most used programming languages today



What we do at LSD

Demonstrations

Sets

Functions

Properties of Functions

Functions in Programming

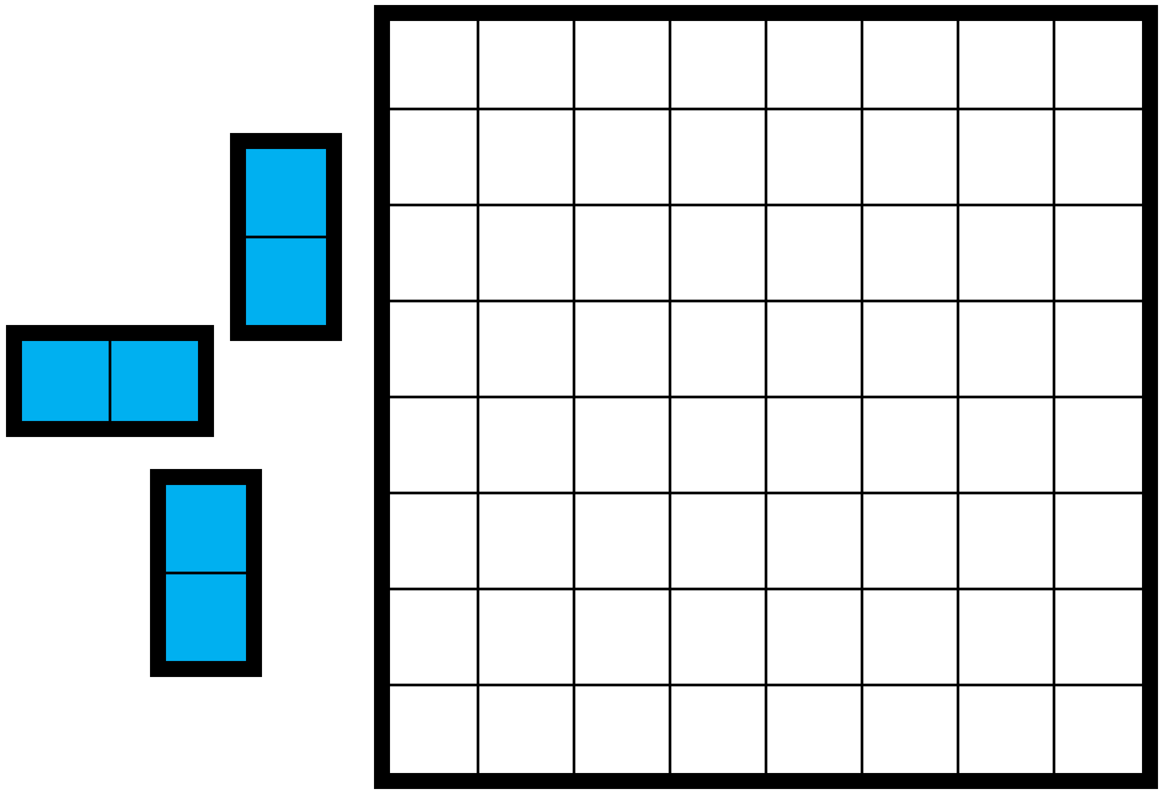
Let's begin

- DEMONSTRATION

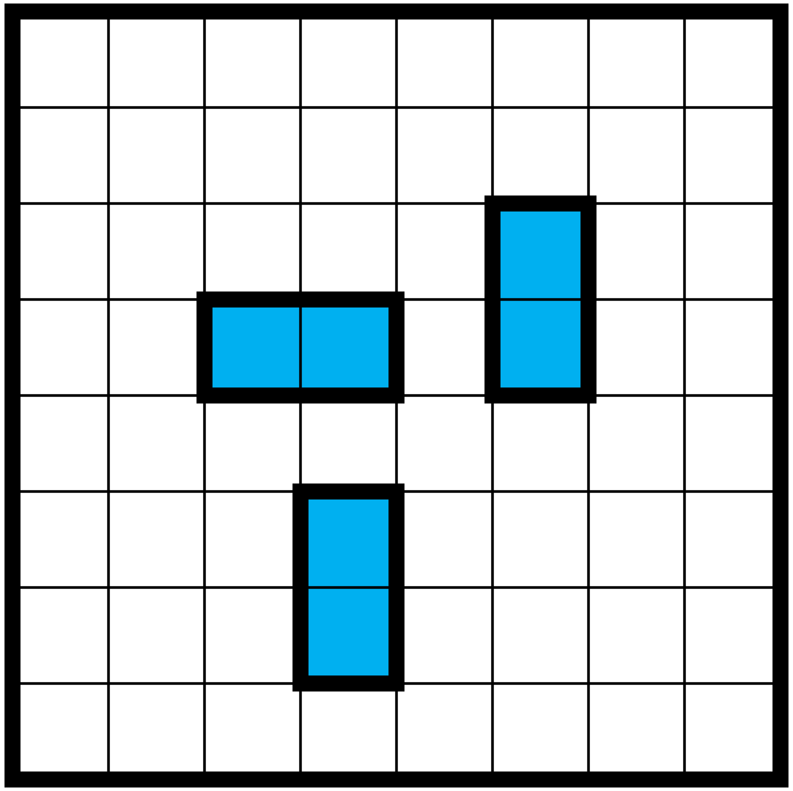
- What is the demo?
  - An argument that is so convincing that you can use it to convince others
  - It's a sign of understanding

DEMONSTRATION

A simple first example: Can we fill (without leaving empty spaces) an 8 x 8 chessboard with 1 x 2 dominoes?

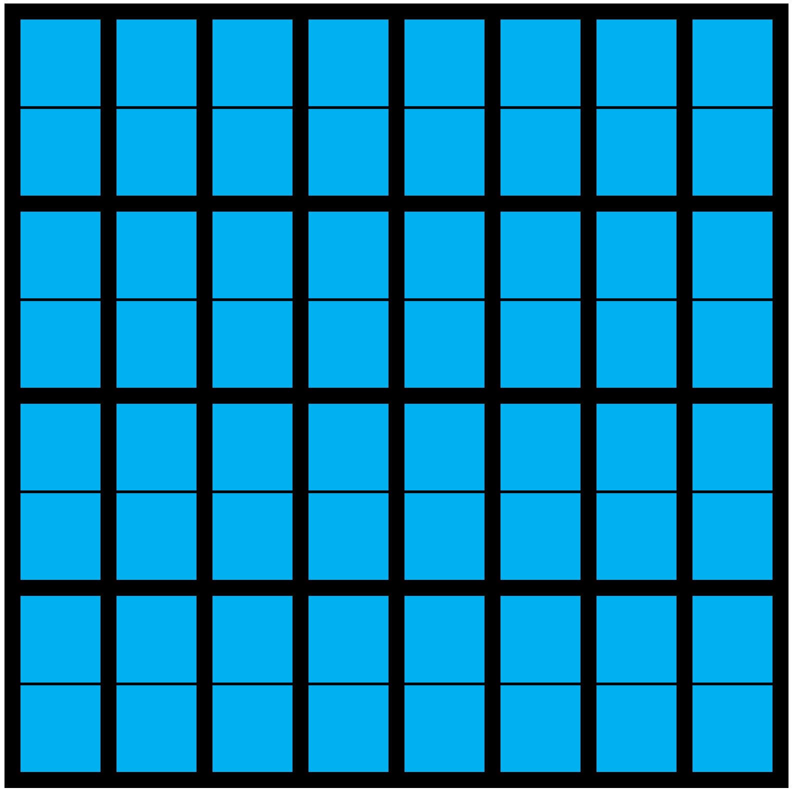


Can we fill it or not?



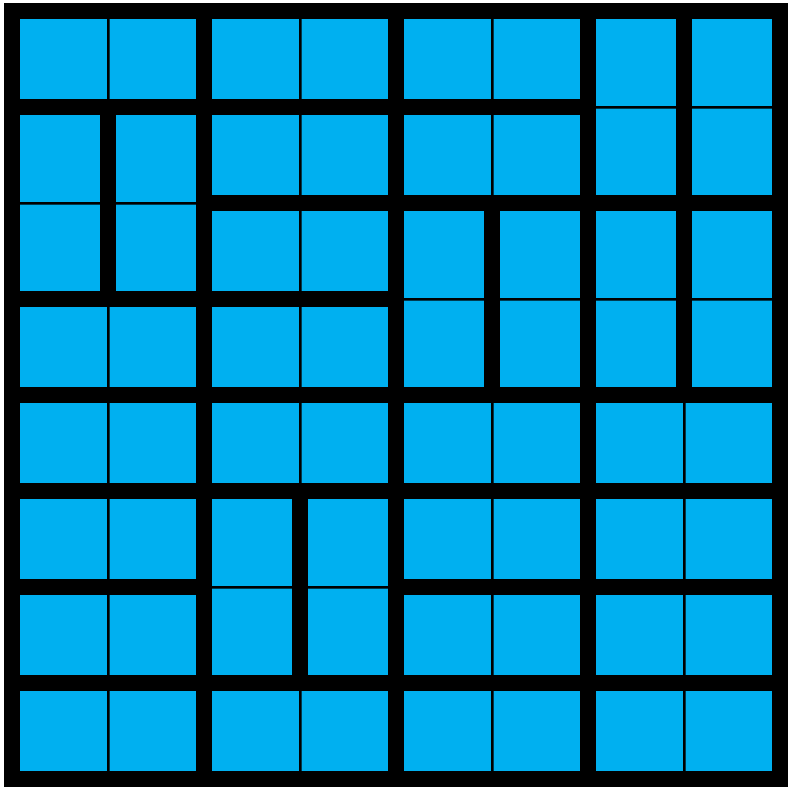
DEMONSTRATION BY EXAMPLE

The answer is obvious, YES. And I think each of you can prove it. It is enough to show an example and we proved:



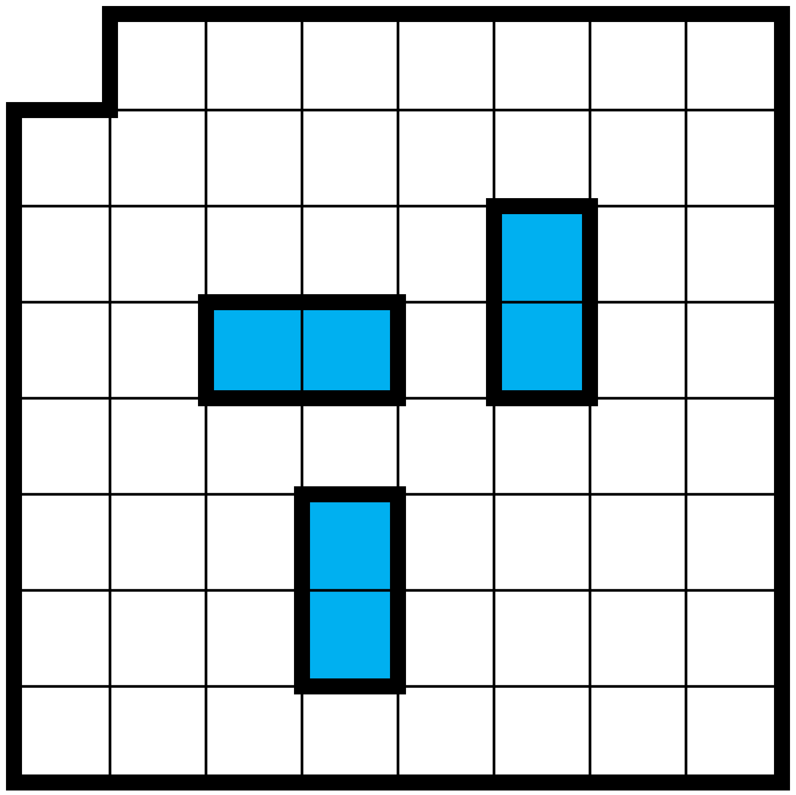
DEMONSTRATION BY EXAMPLE

The answer is obvious, YES. And I think each of you can prove it. It is enough to show an example and we proved:



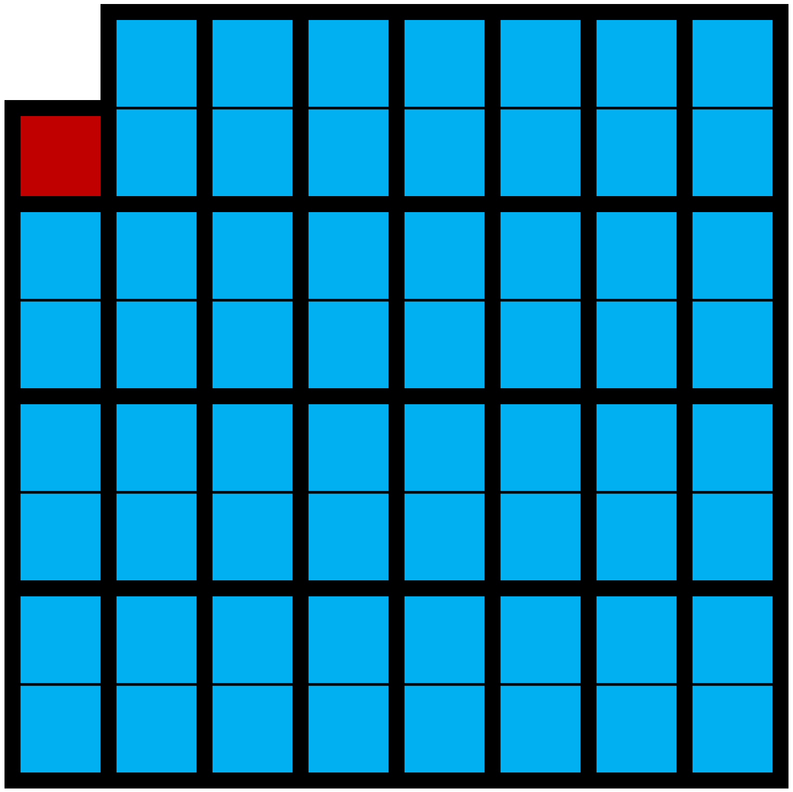
Can we fill it or not?

But if we complicate the matter a bit? What if we remove a square from the board? Can we fill the board again this time?



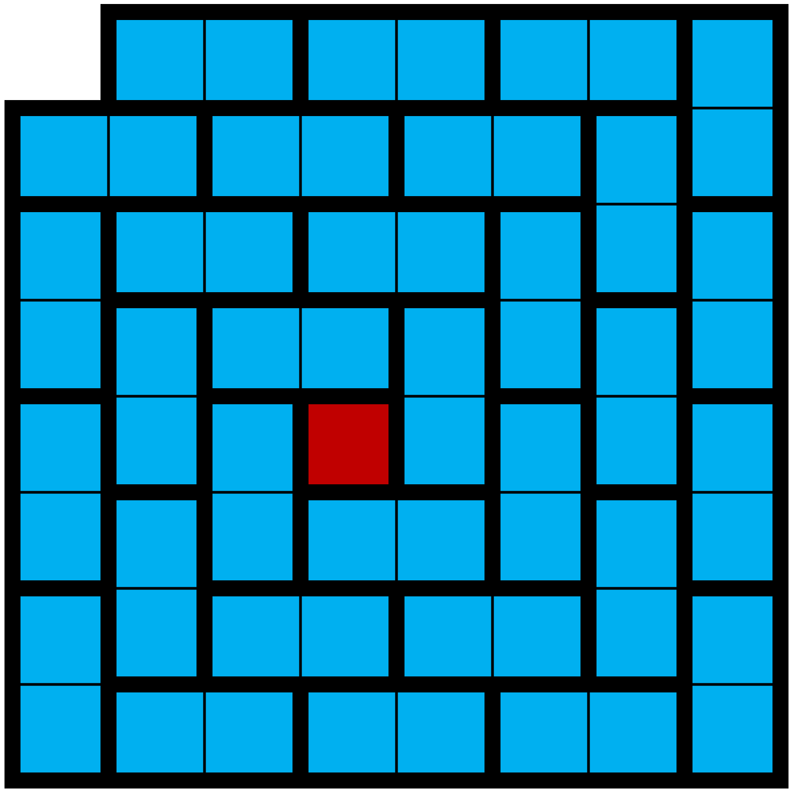
Can we fill it or not?

I have tried 2 ways below but in both we are left with an unfilled square. This time we cannot demonstrate by example . But we can prove that there is no possible combination , can we?



Can we fill it or not?

I have tried 2 ways below but in both we are left with an unfilled square. This time we cannot demonstrate by example . But we can prove that there is no possible combination , can we?



Can we fill it or not?

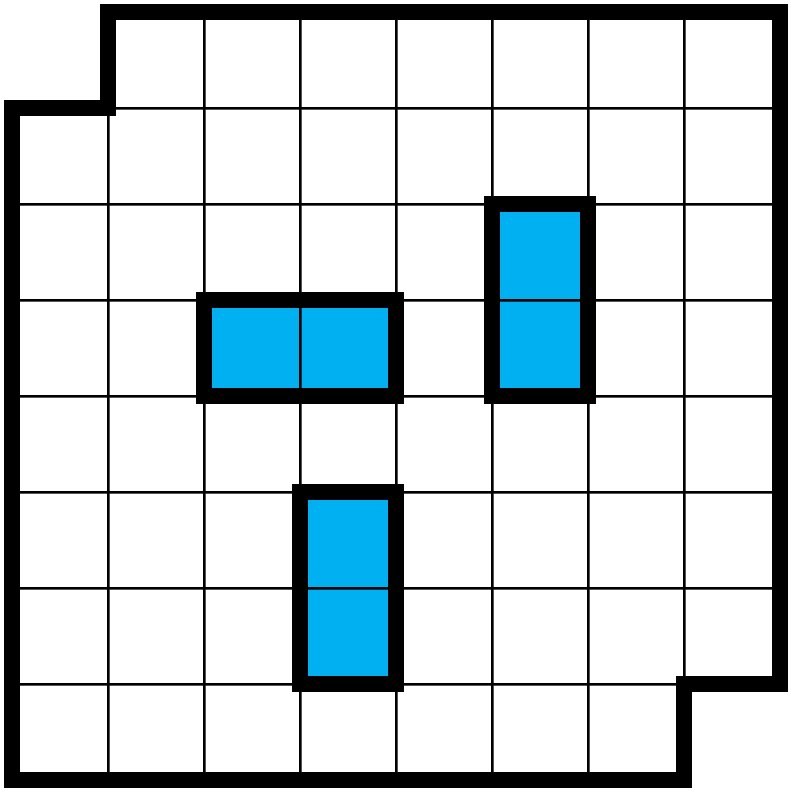
We can prove it like this:

Out of 64 squares on the chessboard (8 x 8). If we eliminate one, we are left with only 63 (8 x 8 – 1).

We can fill with pieces of dominos only an even number of squares (1 x 2) . Because 63 is an odd number, it cannot be covered with the available pieces.

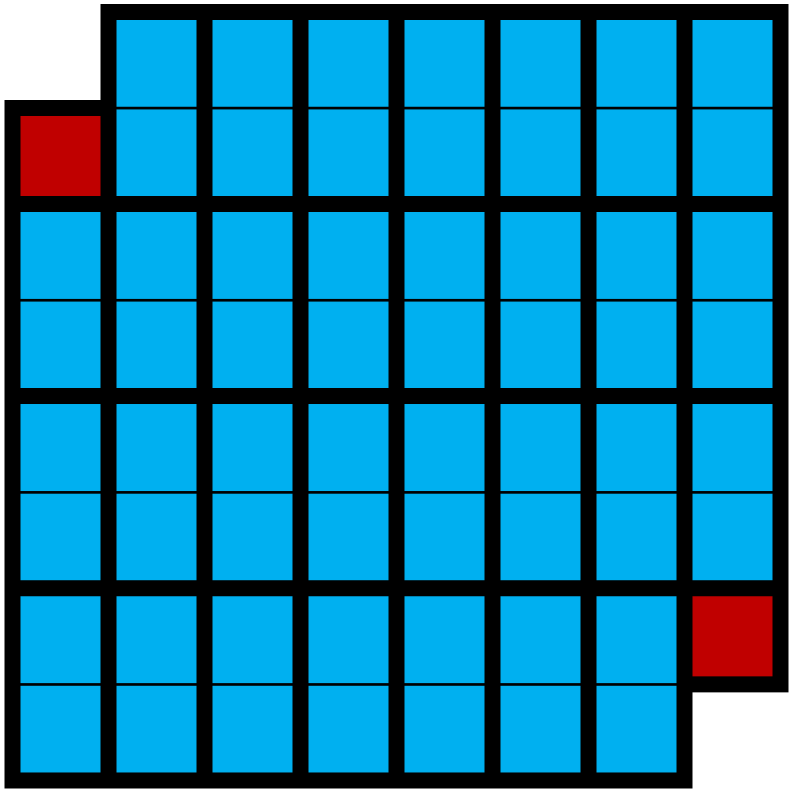
QED

But what if we complicate things a little more? What if we remove two squares from the board? Can we fill the board this time? We only have 62 (8 x 8 -2) squares left to fill, so we could use 31 dominoes, right?



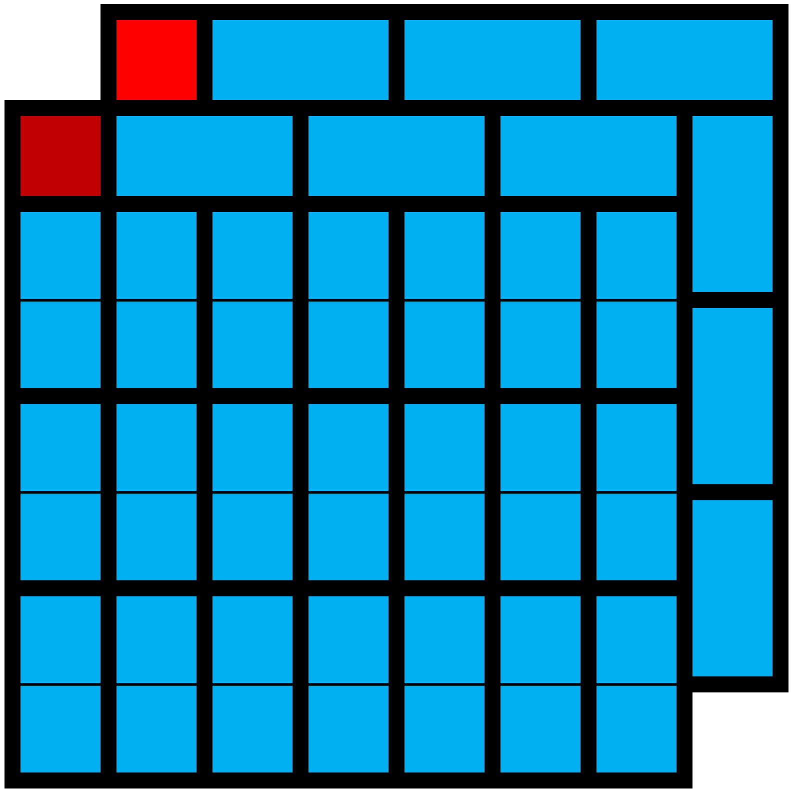
Can we fill it or not?

I have tried 2 ways below, but in both we are left with 2 unfilled squares. Even this time we cannot demonstrate by example . Can we prove, again, that there is no possible combination?

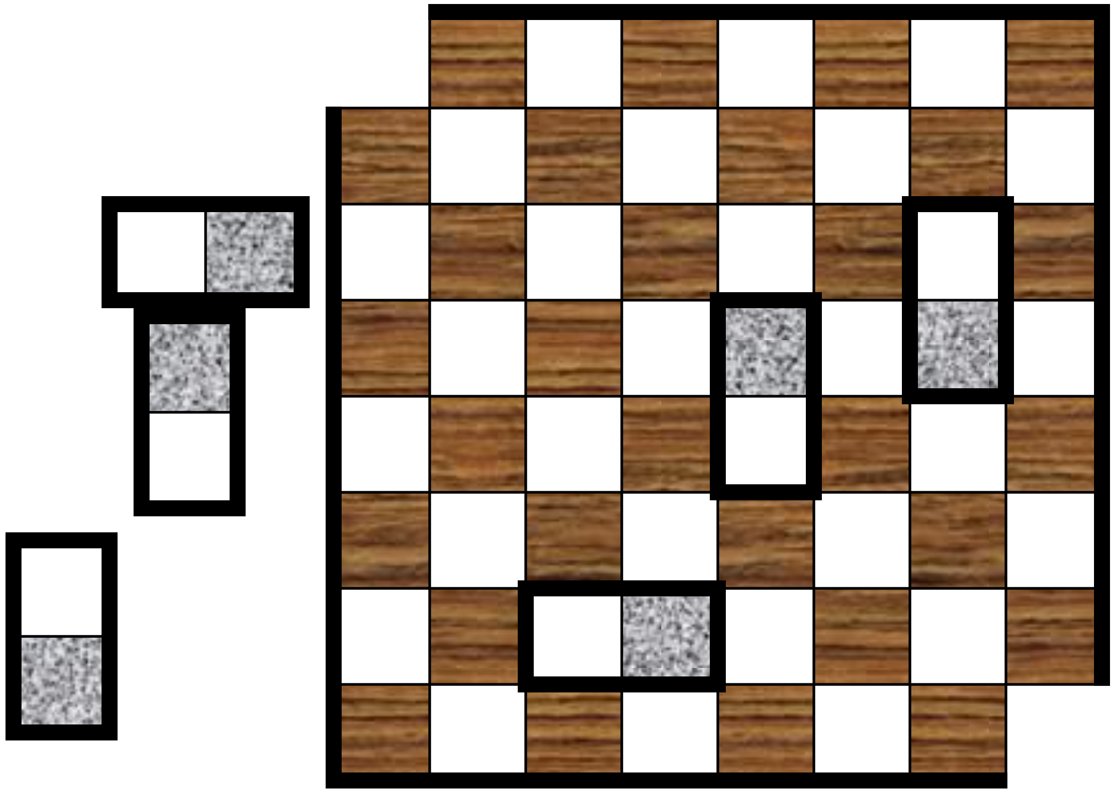


Can we fill it or not?

I have tried 2 ways below, but in both we are left with 2 unfilled squares. Even this time we cannot demonstrate by example . Can we prove, again, that there is no possible combination?



To simplify our demonstration we will use the 2 colors of the chessboard. Each time there will be 2 different colors covered. We removed 2 white squares, so there are 32 brown squares and only 30 white ones left on the board , a surface impossible to cover, because 2 uncovered colored squares will always remain covered.



- Will this demonstration convince you?

- We can continues with the examples. If we remove 2 squares of different colors, can we cover the rest of the board with dominoes?
- Can we prove that we can cover the rest of the board regardless of the 2 different colored squares removed?

What we've learned so far about the demos:

- To prove that something exists, it is enough to give an example
- To prove that something does not exist we have to follow a logical reasoning

We will continue with an example closer to mathematics:

Is there a number?

- Is there a 2-digit number that if it becomes 7 times smaller then its first digit disappears?
- Yes, 3 5 / 7 = 5
- It's not hard to find, the 2-digit numbers divisible by 7 are just a few: 14, 21, 28, 35 , 42, 49, 56, 63
- But we can complicate the problem: Is there a number that, if it becomes 57 times smaller, then the first digit disappears?
- 7 125 / 57 = 125

- Is there a number that, if it becomes 57 times smaller, then the first digit disappears?
- 7 125 / 57 = 125
- How can we find it mathematically:
  - We write down the searched number with ab...z, where each letter is a number
  - ab...z = 57 * b...z
  - X = b...z we consider to have k digits
  - a * 10^k + X = 57 * X
  - a *10^k = 57 * X – X = 56* X = 7 * 8 * X
  - a *$5^k * 2^k$ = 7 * 8 * X
  - a must be divisible by 7, a has 1 digit, so a = 7
  - 7 *$5^k * 2^k$ = 7 * 8 * X
  - $5^k * 2^k$ = 8 * X
  - $5^k * 2^k$ = 2^3*X
  - If k = 3 we have the solution X = 125
  - So we found a number of the searched 7125 = 57*125

Types of demonstrations

- By example - we have seen so far
- Proof by contradiction - we will go through
- By mathematical induction – we will go through

Proof by contradiction

The contrapositive of a statement

- Example:
  - The sentence "If it rains implies that I take my umbrella."

  is equivalent to its contrapositive :
  - "If I don't take my umbrella it means it's not raining."

- In logic, the above example is formally transposed as follows:
  - We write the sentence "It's raining" with P
  - We mark with Q the sentence "I take my umbrella"
  - "If it's raining, it means I'm taking my umbrella." is transcribed $P \Rightarrow Q$, where P is the premise and Q is the conclusion
  - $\neg$ P and $\neg Q$ are the negations of the 2 sentences
  - a sentance is equivalent to its contrapositive:

$$P \Rightarrow Q \qquad \Leftrightarrow \qquad \neg Q \Rightarrow \neg \mathrm{P}$$

Proof by contradiction

- The proof by contradiction uses the equivalence between a statement and its contrapositive:

  $P \Rightarrow Q \qquad\qquad \Leftrightarrow \qquad\qquad \neg Q \Rightarrow \neg \mathrm{P}$

  Statement the contrapositive

  – suppose the false conclusion ($\neg Q$)

  – We show then the premise is false ($\neg$P) , which is absurd, knowing that the premise is true (P)

  – the conclusion cannot be false $\Rightarrow$ the conclusion is true

Proof by contradiction

- Example: The sum of three natural numbers is 139. Prove that at least one of them is higher than 46 .

  - *P* : *"a+b+c = 139, where a, b, c are natural numbers"*
  - *Q* : *"a > 46 or b > 46 or c > 46"*
  - *build* $\neg Q$: *" a ≤ 46 and b ≤ 46 and c ≤ 46 "*
  - $\neg Q$ ⇒*" a+b+c ≤ 46+46+46 ≤ 138 "* contradicts P which says that the sum is 139 ⇒ ¬P
  - Therefore P ⇒ Q is true
  - QED

Proof by mathematical induction

- If a sentence P (n) depends on a natural number n and

  1) base case: if P(1) is true

  2) the inductive step: for any $n \geq 1$

  $P(n) \Rightarrow P(n + 1)$

- then P(n) is true for any n.

Proof by mathematical induction

- Example: Prove that

$$9^n - 1 \vdots 8, \forall n \in \mathbb{N}*$$

1) we calculate P(1)= $9^1 - 1 = 8 \vdots 8$ - true

2) assume P(n) true

$$P(n) = 9^n - 1 \vdots 8$$

calculate $P(n+1) = 9^{n+1} - 1$

$= 9^n * 9 - 1$

$= 9^n * 9 - (9 - 8)$

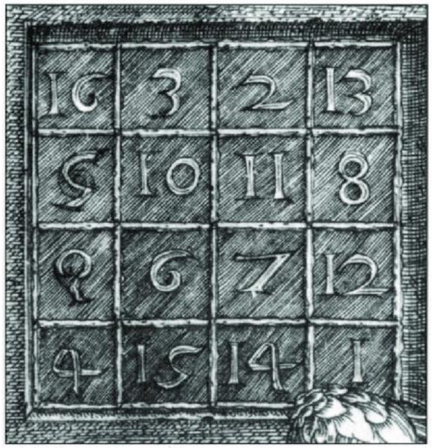
$$= 9 * (9^n - 1) + 8$$

$$= 9 * P(n) + 8$$

$P(n) \vdots 8 \text{ și } 8 \vdots 8 \Rightarrow P(n+1) = 9 * P(n) + 8 \vdots 8$ – *true*

$9^n - 1 \vdots 8, \forall n \in \mathbb{N}*$

QED



What we do at LSD

Demonstrations

Sets

Functions

Properties of Functions

Functions in Programming

Introduction to Sets

B426 B528a
B528b
B418a

What are the sets

Definition:

A set is a collection of objects that are called elements (of the set).

We have two distincts notions: elements and sets

- $x \in S$: element x is an element of S
- $y \notin S$: element y is not an element of S

The order of elements does not matter $\{1, 2, 3\} = \{2, 1, 3\}$

One element can not appear many times $\{1, 2, 3, 2\}$

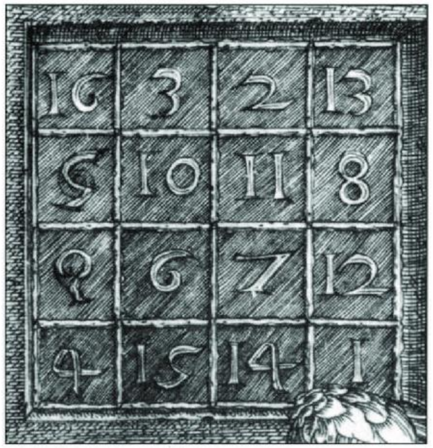
Subsets

A is *a subset* of *B* : $A \subseteq B$

if every element of *A* is also an element of *B* .

To *prove* A $\nsubseteq B$ it is enough to find an element $x \in A$ for which $x \notin B$.

- If $A \subseteq B$ and $B \subseteq A$, then $A = B$ (the sets are equal)



What we do at LSD

Demonstrations

Sets

Functions

Properties of Functions

Functions in Programming

FUNCTIONS

A function from a set A into a set B is a relation from A into B such that each element of A is related to exactly one element of the set B.

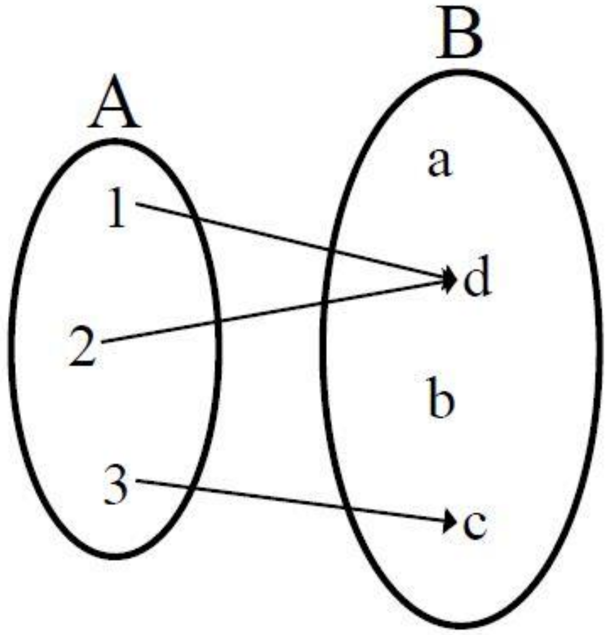


Image: http://en.wikipedia.org/wiki/File:Total_function.svg

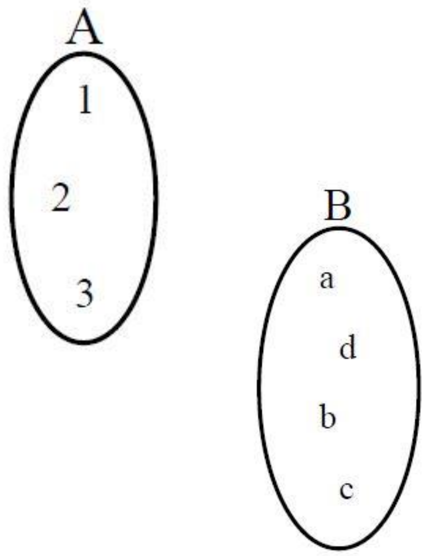
A function has 3 components

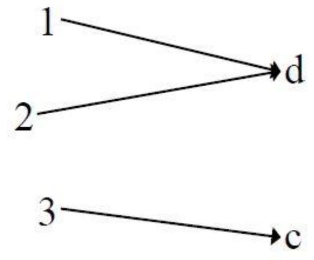
1. *the domain of definition*

2. *value domain (* codomain)

3. *the actual association*

(law, rule of association, correspondence)

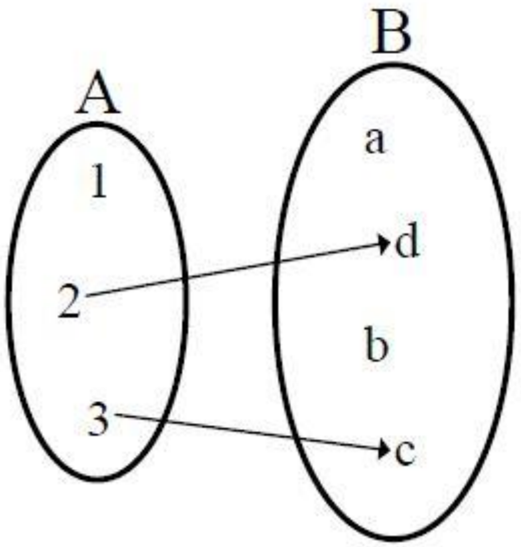


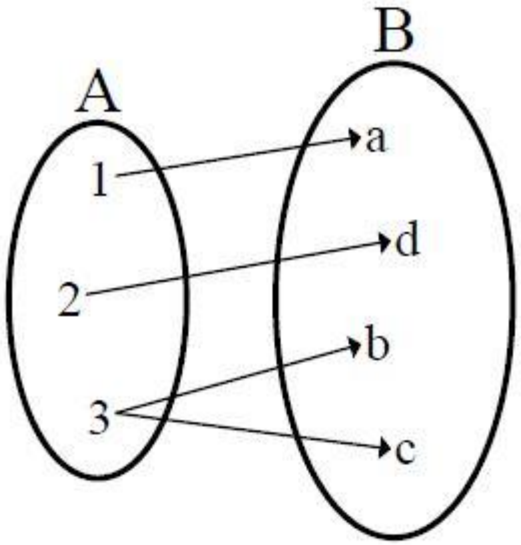


$f : Z \to Z, f(x) = x + 1$ and
$f : R \to R, f(x) = x + 1$
they are distinct functions !

Image: http://en.wikipedia.org/wiki/File:Total_function.svg

Examples that are not functions

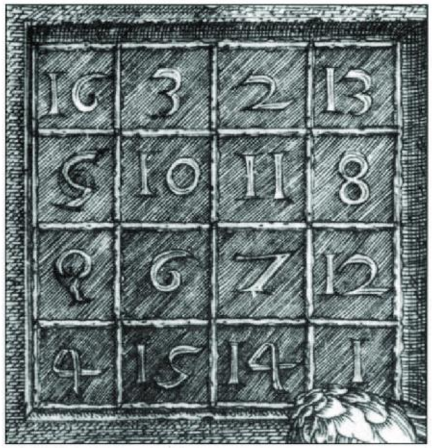




associates multiple values to an element

It don't associate a value for each elements from A

Image : http://en.wikipedia.org/wiki/File:Partial_function.svg _ http://en.wikipedia.org/wiki/File:Multivalued_function.svg



What we do at LSD

Demonstrations

Sets

Functions

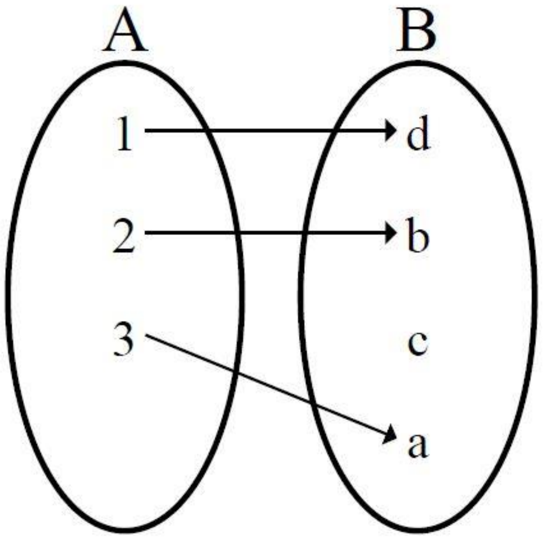
Properties of functions

Functions in programming

Injective functions

A function $f : A \rightarrow B$ is *injective* if for any
$x_1, x_2 \in A, x_1 \neq x2 \Rightarrow f(x_1) \neq f(x_2)$
(associate *different values* to *different arguments*)

injective function



non-injective function

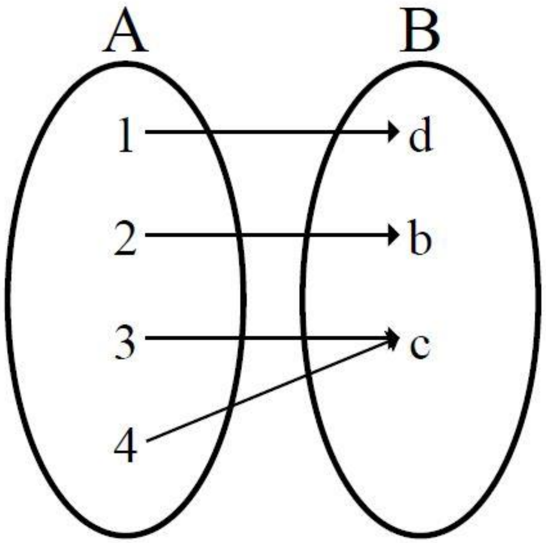


Image: http://en.wikipedia.org/wiki/File:Injection.svg
http://en.wikipedia.org/wiki/File:Surjection.svg

Injective functions

Instead of the condition

$x_1, x_2 \in A, x_1 \neq x2 \Rightarrow f(x_1) \neq f(x_2)$

we can write equivalently :

$$f(x_1) = f(x_2) \Rightarrow x_1 = x_2$$

(if the values are equal, then the arguments are equal)

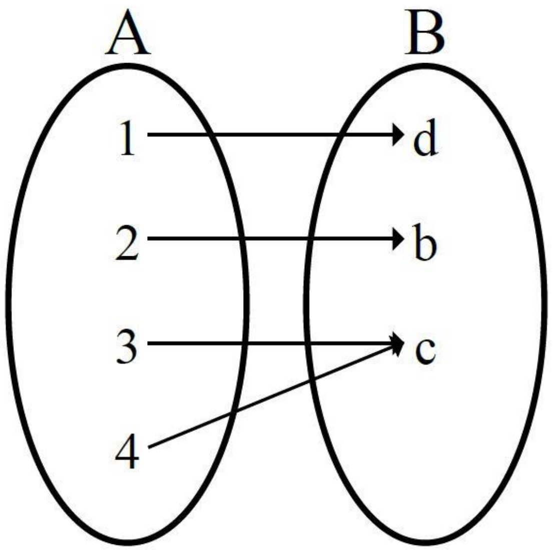
Injective functions

Property of injective functions:

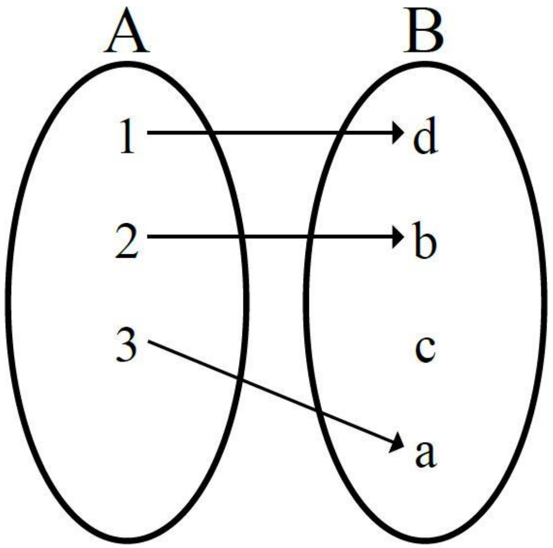
If $f : A \rightarrow B$ and f is injective , then / A / $\leq$ $/B/$.

Surjective functions

A function $f : A \to B$ is *surjective* if for each $y \in B$ exists an $x \in A$ with $f(x) = y$.



surjective function



nonsurjective

Image: http://en.wikipedia.org/wiki/File:Surjection.svg
Image: http://en.wikipedia.org/wiki/File:Injection.svg

Surjective functions

Property of surjective functions:

If $f : A \rightarrow B$ and f is surjective , then $|A| \geq |B|$.

We can transform a non-surjective function into a surjective one by *restricting* value range:

- $f_1 : \mathrm{R} \rightarrow \mathrm{R}, f_1(x) = x^2$ is not surjective ,
- but $f_2 : \mathrm{R} \rightarrow [0, \infty), f_2(x) = x^2$ (restricted to non-negative values) is surjective .

Bijective functions

A function that is injective and surjective is called *a bijective* .

A bijective function $f : A \to B$ matches one *to one* the elements of A with the elemetnts of B .

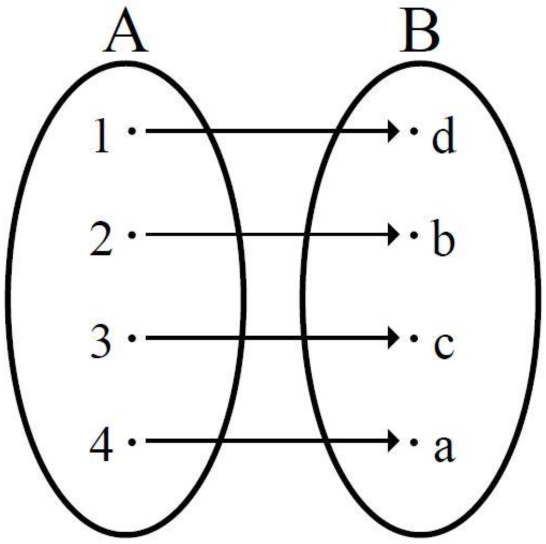
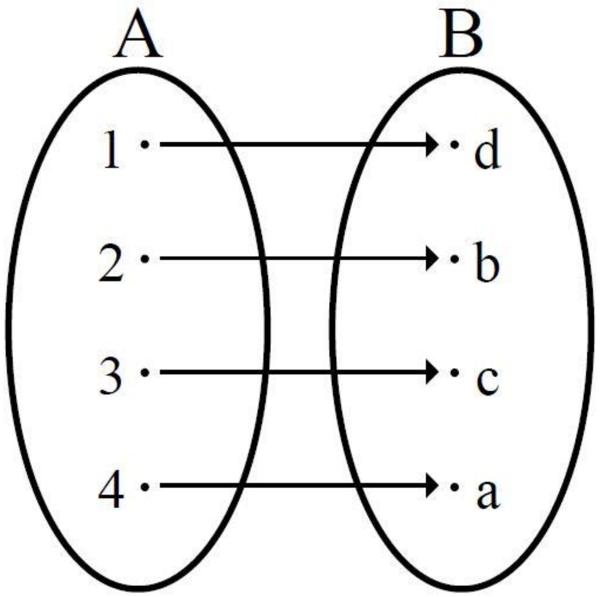


Image: http://en.wikipedia.org/wiki/File:Bijection.svg

Bijective functions

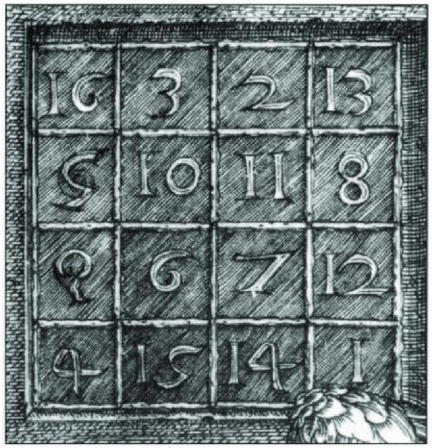
For *any* function, from the definition , to each $x \in A$ there corresponds a *unique* $y \in B$ with $f(x) = y$.

For a *bijective* function , and vice versa: to each $y \in B$ there corresponds a *unique* $x \in A$ with $f(x) = y$.



If there exists $f : A \to B$ and f is bijective, then $|A| = |B|$.

Image: http://en.wikipedia.org/wiki/File:Bijection.svg


16 3 2 13
5 10 11 8
9 6 7 12
4 15 14 1

Function in programming

In programming languages, a function expresses a *calculation* : it receives a *value* (*the argument*) and produces as a *result* another *value.*

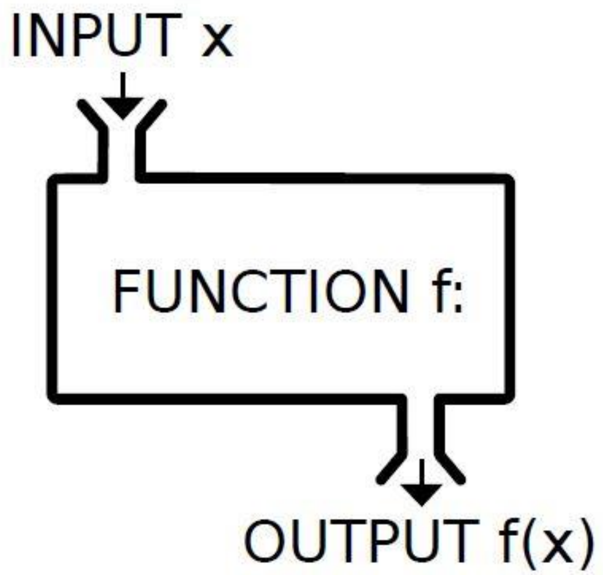


Image: http://en.wikipedia.org/wiki/File:Function_machine2.svg

Functions in Python

Functions are defined simply, with a syntax (writing rule) similar to other programming languages. Thus, the function

$f : \mathbf{Z} \rightarrow \mathbf{Z}$, $f(x) = x + 3$ is written in Python :

```
def f(x):
        return x + 3
```

The def keyword introduces a *definition* , here, for the identifier f .
After the name of the function, enter its parameters between brackets (*x*) , followed by the sign **:** (colon).

Functions in Python

```
def f(x):
        return x + 3
```

In Python, *indentation* is very important in writing code.

Unlike other programming languages that use braces { }, in Python indentation is used for blocks of code.

Indentation is always preceded by the sign : (colon).

Calling Functions in Python

Once the function is defined, it is called as follows:

```
>>> f( 1 )
4
```

When we call a function we can also specify the name of the parameter at the call :

```
>>> f(x=5)
8
```

We can also give the function a complex expression as a parameter:

```
>>> f(2*3)
9
```

Anonymous functions in Python

Notation lambda *argument* : *expression* defines in Python an anonymous function (lambda function).

This is a function *expression and can be* used in other expressions. We can directly evaluate :

```
>>> ( lambda x : x + 3 )( 2 )
5
```

without having to give the function a name first.

This simple example illustrates that in Python, a function can be used just as easily as any other value.

Anonymous functions in Python

Using the notation def , it is equivalent to we define :

```
>>> def f(x):
...     return x + 3
```

to have

```
>>> def f(x):
...     return ( lambda x : x + 3 )(x)
```

Functions with multiple arguments in Python

Let the function in mathematics be :

$$amount : \mathbf{Z} \times \mathbf{Z} \rightarrow \mathbf{Z}\ ,\ sum\ (x\ ,\ y) = x + y$$

One way to write this function in a Python program is :

```
>>> def sum (x, y):
...       return x + y
```

Functions are values in Python

A function is also a value (like integers , reals , etc.) and can be used just like any value (such as a parameter, returned , etc.) :

```
>>> def g ( f, x):
...       return f( x ) + x
```

Case-defined functions in Python

$$abs : Z \to Z,\ abs\,(\,x\,) = \begin{cases} x,\ \mathit{if}\ x \geq 0 \\ -x,\ \text{if}\ (x < 0) \end{cases}$$

The value of the function is not given by a single expression, but by one of two different expressions depending on a condition ($x \geq 0$).

In Python:

```
def abs(x):
        if (x > 0):
                return x
        else:
                return -x
```

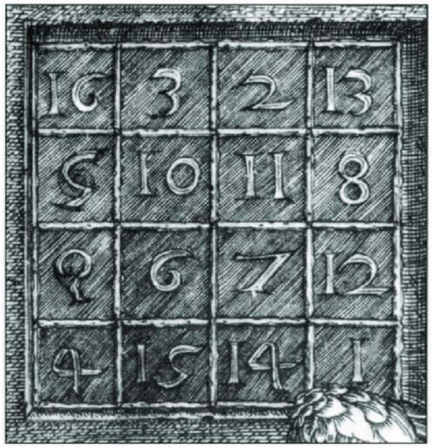
Case-defined functions in Python

The general case of the *if statement is:*

```
if ( expr1 ):
        express2
else:
        express3
```

The if statement is evaluated like this:

- If the evaluation to expr1 gives the value “true” , the final value of the expression is the value of expr2, otherwise (if expr1 is “false”) it is the value of expr3.



Have a good day!

Bibliography

- The checkerboard and domino pieces examples were inspired by ***the Mathematical Thinking in Computer Science course*** from University of California San Diego (from https://www.coursera.org/)
- 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)