Training Plan for After School Sessions

Co-funded by the European Union

Funded by the European Union. However, the views and opinions expressed are solely those of the author(s) and do not necessarily reflect those of the European Union or OeAD-GmbH. Neither the European Union nor OeAD-GmbH can be held responsible for them.

TOPICS & Games

1. Introduction

Transfer of the Treaty with Orsi

→ we add our own contract/rules in IT

You have to get it:
- Mail from
- Packaging paper
- Kaja
- pencils/fountain pens/pencils

Dating

Introducing each other in groups of 3
- Who is who
- What are you doing here?
- What does social media (FB,TikTok, Insta, BeReal) mean to you?

What is IT?

This unit is designed both to give children an overview of what will happen in IT during the school year, and also to allow us to assess the level of interest in IT.

Description of the academic year

Unit	Aim	Content
1	What is IT? School year brochure	1. About 2. Fill in the questionnaire (online) 3. Guide to the academic year 4. What is IT? a. What do you know? b. Where does it come from, what do we use it for?
2	PROG BASICS (variables, types of variables, display description, arithmetic operations)	1. What are the variables? What we use them for 2. Description of variables 3. Game(s) 4. C# programming
3	PROG BASICS (variables, display description)	1. Repeat variables 2. What are the classes 3. Game(s) 4. C# programming

4	Logic - What is logical? - Sentences that make sense (separately, together) - True false - AND OR NOT	1. WHAT is the logic? 2. Logical sentences 3. AND OR NOT 4. Which is logical, right?
5	Logic	1. Reproduction 2. In C#, the boolean 3. Arithmetic & Boolean in C#
6	ALgorithms	1. What are the algorithms? 2. What are the elements? 3. Examples/Tasks 4. Game
7	Algorithms & Introduction to Loops and Statements	1. Algorithms for repetition 2. Algorithms & Logics 3. Game
8	IF ELSE, (DO) WHILE, FOR (EACH), switch case	IF ELSE 1. Game repetition 2. Processing 3. C#
9	IF ELSE, (DO) WHILE, FOR (EACH), switch case	FOR 1. Game 2. c#
10	IF ELSE DO WHILE FOR SWITCH CASE	While + DO While 1. Game 2. c#
11	UNITY	1. Getting to know the programme a. Installation b. Register at c. get to know
12	Getting to know Unity + RANDOM	1. When can randomness occur? 2. Display a coin //TODO Display a prototype coin 4. Coin falls on random side
13	RANDOM + COIN FLIP	1. Random in nature? 2. Coin falls randomly? 3. Coin flip 100x and write 4. Which side does it fall on?
14	COIN FLIP + VAL NUMBER	1. Draw a flowchart

		2. Programming coin flips in C# //TODO protoipus 3. Val number funds
15	VALSZÁM + KOCKA/Lift inhaler (Combinatorics)	1. Value number + cube 2. Cube throwing 3. C# cube programming //TODO prototype
16	PROG HALADO (data, classes, methods, data structure)	First session of the 2nd semester: filling in a questionnaire! The essence of PROG HALADO: 1. Reproductions from 2. Filling gaps A playful approach to repeated processing of games Reprogramming things in C#
17	PROG HALADO (data, classes, methods, data structure)	What is data, how exactly is it built up? What are the types (small repetition)
18	PROG HALADO (data, classes, methods, data structure)	What are the classes? How are they structured? How do we recognise them, what are they for? How-where do we use them?
19	PROG HALADO (data, classes, methods, data structure)	More complex data structures (e.g. trees, graphs) More for interest than to teach everything but important to understand when is it ideal to use them and why?
20	PROG HALADO (data, classes, methods, data structure)	Repetition of the above
21	VALSZÁM + KOCKA/Lift inhaler (Combinatorics)	Unity → coin flip integration in unity
22	GAME PLANNING - DENSION FA	Making rules, demoting, evaluating, modifying if necessary Talking about cyclical thinking - a bit "Frame numbers" how many cards How many rounds How many castes How many aspects How many points
23	GAME PLANNING - Rules - 4-6 aspects 100 points scattered	Castes & facets design help with EE: 1. comfort zone - superpower from there? 2. Which side are you on? Which set are you on?

	- Castes (how many, what kind)	3. Processing
24	GAME PLANNING	Cyclicality rules, creation, repetition
25	Game design	Cyclicality rules, creation, repetition
26	RESET & Repeat	PROTOTYPE PROGRAMMING: Required Define classes in C#, represent them in unity
27		Definition of departmental responsibilities
28		Implementation of classroom functions
29		Connecting the game loop, adding cards to the program
30	ANIMATION	Depositing a card
31	ANIMATION	Swipe card
32	...	Animating a deck that is about to expire
33	Develop a card game (should be tangible) Skills required - Branching loops (if else, for while, do while, switch case, ...) - Knowledge of object-oriented programming (classes, methods, variables, scopes of variables, polymorphism) - Variable types - Basic concepts: data , matrix, attached data file, data structure - Arithmetic operations - randomization	

What is IT?

The aim of this block is to assess who knows what about IT. We ask them questions to make them aware of the wide presence of IT, the fact that they use the phone every day, use the internet, buy a sandwich in the shop... etc.

- What do you think IT is?
- What are they used for?
- What do you use it for?

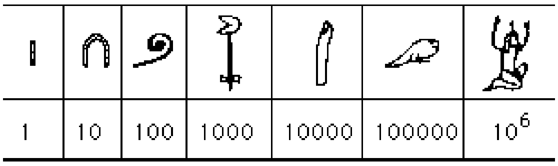
- What games do they know?
- What do you learn about IT at school?
- What do you think an IT specialist does?
- Where is IT used and for what? (examples)
    - High-precision tasks: car manufacturing, medicine, aerospace.
    - High volume, mechanically repetitive tasks: factories.
    - Storing and processing large amounts of data: social security, banks, public administration.
    - Systems that require constant availability: transport, water and electricity, telephone networks, media.
    - Design: for architectural, mechanical and other engineering design, medicine.
    - Other areas: computer graphics, cinema: film tricks, culture, education.
    - School - computer science lessons
    - Shops - Barcode reader
    - military informatics;[14]
    - building informatics
    - commercial information technology;[16]
    - business information technology;[17]
    - health informatics[18]
    - statistical informatics;[19]
    - medical informatics;[20]
    - legal informatics;[21]
    - administrative informatics;[22]
    - forensic informatics;[23]
    - geoinformatics (geospatial informatics);[24]
    - bioinformatics;[25]
    - physical informatics[26]
    - astronomical informatics[27]
    - chemical informatics (cheminformatics, information chemistry)[28]
    - sport informatics

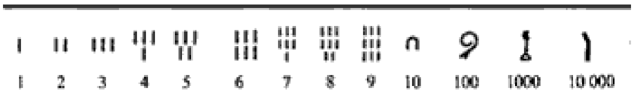
- What does a programmer do that you use?
    - website development
    - Robots
    - Games
    - Cash register programme
    - tr.

The history of information technology

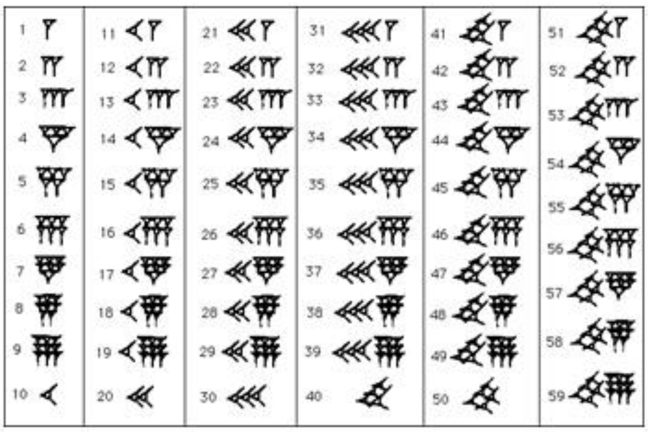
The history of computing is identified with the history of computational tools, which is practically the same age as the origin of man. To count, prehistoric man first used his fingers, later stones and pieces of thread, and recorded the results by carving them into the walls of caves, bones or wooden tablets.

- In Egypt, by around 2000 BC, the decimal number system was well established and its spread was driven by the needs of agriculture and astronomy. A separate symbol was used for each higher decimal unit, so the concept of local value was not yet known. One stick: 1; two sticks: 2; three sticks: 3; and so on. One hairpin: 10; two hairpins: 20; three hairpins: 30; and so on. One screw: 100; two screws: 200; three screws: 300; and so on. One lotus flower: 1,000; two lotus flowers: 2,000, and so on, up to millions. The million was expressed by the kneeling, wondering human figure. These signs are shown in the figure below:





- In Mesopotamia, in the late Sumerian period, where canalisation and construction required complex calculations, we find an advanced local valuable hexadecimal number system, which still reveals the use of the previous decimal number system, since the digits from 1 to 60 were written using the older decimal number system. Their cuneiform markings were stamped in clay, the clay tablet was fired in a fiery furnace, making it so timeless that one only has to find the tablet, decipher the secret of the cuneiform writing and it will remain legible until the end of time. An image of a clay tablet and the cuneiform equivalents of the numbers are shown in the diagram below:



- The ancient Hungarians used the decimal number system in historical times. This was, however, the result of a long development over the six and seven number systems of earlier times. The seven number system can be inferred, for example, from the seven-headed dragon of fairy tales, the seven-seven country, the seven-length beard, the seven-mile boot, the seven-secret secret. In our later numerals eight and nine, the final c, or z as it used to be spelt, is probably the ending of the numeral ten. From this we suspect that eight and nine were derived from ten by our ancestors. According to linguistic research, two, three, four, five, six and one hundred have a common root in the Finno-Ugric languages. At the time of their development, the Finno-Ugric peoples were still together and used a six number system. The number seven is a reference to the narrower Ugric family (Hungarian, Vogul, osztják). In the languages of these peoples, the word seven is not only a number but also a duration of seven days. The digits of the ancient runic script and the way they are written (see the following diagram) refer to the decimal number system. Not only the shape of the numbers, but also their names were taken from the Indians. Later on, the Arabic numerals were constantly changing.

٠	١	٢	٣	٤	٥	٦	٧	٨	٩
0	1	2	3	4	5	6	7	8	9

Later, with the advent of larger numerical values, number systems evolved and tools to help counting appeared. The first such tool was the abacus (also known as the soroban). According to historians, the abacus was the first counting instrument, and its ancient forms are found in almost all ancient cultures. It usually consists of a few thin rods/rods, each with a specific number of sliding discs or balls, possibly of different colours. These are used by the operator to perform addition and subtraction (multiplication and division can only be performed with complex rules).
E.g. addition with abacus:
https://www.youtube.com/watch?v=djrd2f63Oy4&ab_channel=RitaG%C3%B6nczi

The first - mass-produced - calculator was made between 1642 and 1644 by Blaise Pascal (1623-1662),

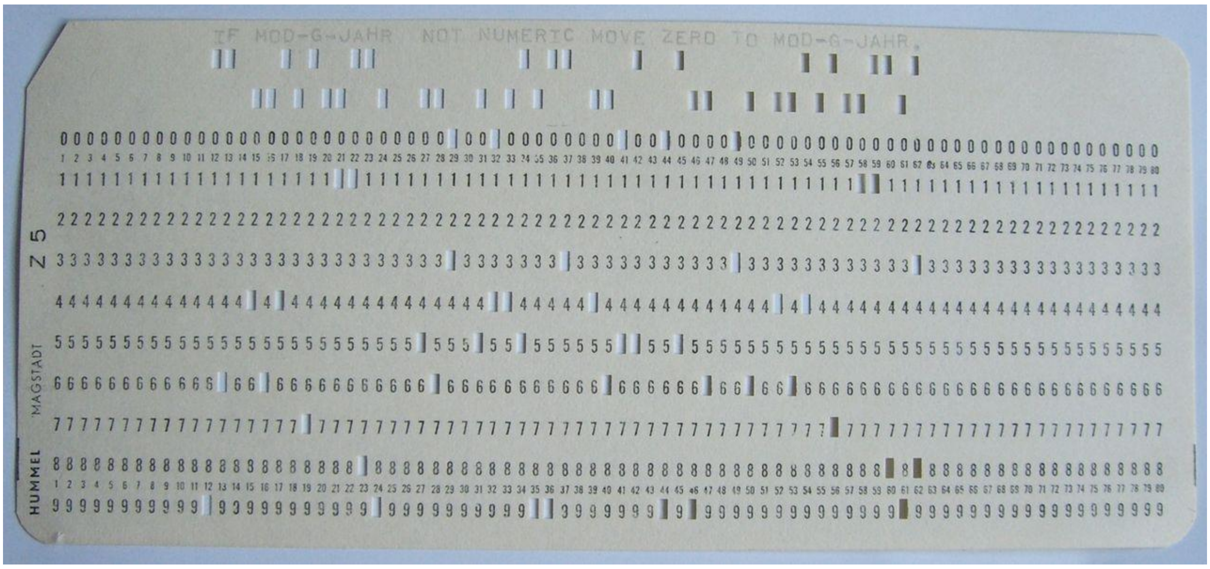
Pascal's calculator was further developed by Gottfried Wilhelm Leibniz (1646-1716). This machine was the first to perform division and multiplication directly, and subtraction without any additional operation.

In 1939, Konrad Zuse's first highly successful mechanical system computer, the Z1, was built. It was the first machine based on the binary number system.

t Howard H. Aikent (1900-1973) and IBM signed a joint development agreement in 1939, which resulted in the 1944 Harvard Mark I, an electromechanical machine that could be controlled by a series of instructions written on a paper tape. The machine was about 100 times faster than a good manual calculator, worked non-stop, and completed six months' work in a day.

The operating principles of computers as we know them today were developed by János Neumann (1903-1957), a Hungarian-born scientist, using his experience in military technology. By 24 June 1945, an abstract - First Draft of a Report on the EDVAC - was completed, giving a full analysis of the planned structure of the EDVAC. It included the proposed architecture of the computer, the logic circuits needed to build the components and the code for the machine. Most computers today are still built on the principles set out in the report. His main theorems are now known as the Neumann principles. It was the first electronic, digital, universal computer with internal program control.

All the machines made so far have written their various codes/algorithms on a lock card. Here is an example:



IF statement written in COBOL on a lock card

By the 1950s, the UNIVAC (Universal Automatic Computer) was created on the model of the EDVAC. By 1956, more and more institutes and more and more industries in the USA were developing cathode ray tube computers. By this time, IBM was no longer content with producing punch-card units and printers, but embarked on a computerisation programme that would secure its leadership for at least 50 years. Mass production of computers began
.

2-3. Variables & Classes

What are variables good for?

When you write a program, you may need containers where your data is temporarily stored. These stores are called variables.

Example : gym bags as a storage unit = variable . important unit to go to a sports class, the contents may vary depending on the type of sport (boxing vs yoga) / school year but e.g. a car or a house will not fit.
Just because lunch fits in a gym bag doesn't mean it belongs there.

The concept of variables:
- is used to store or annotate a value (space in memory),
- can be freely modified.

Type of variables:

determines the value that can be stored
- integer (1, 2, 4 bytes): integer, long

- real number (4, 8, 8 byte fixed point): double, float, long
- text (max. 232 bytes): string, char (one character)
- logical variable (2 bytes): - Boolean (true, false TRUE/FALSE)

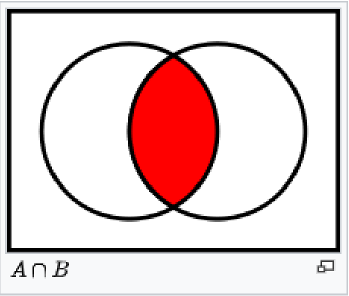
A journey into the world of numbers

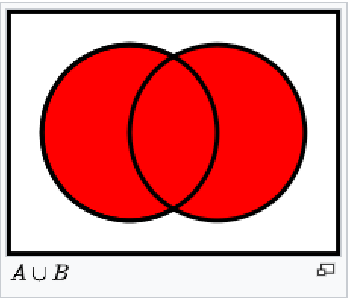
To correctly classify the different numbers for the type of variable, you need to know their properties

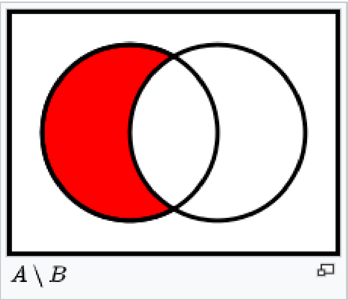
Clusters

Sets - a collection/grouping of objects that are different from each other. E.g. the IT group is made up of children who have different personalities
Empty sets - A set that has no elements is called an **empty set** e.g. living dinosaurs
Subsets - a subset of A is a subset of B if all the elements of A are also elements of B e.g. spaniels are a subset of dogs

Set operations


$A \cap B$


$A \cup B$


$A \setminus B$

Toys:
Dating game
Sections:
Find someone in the room who
- Same eye colour
- Same favourite food
- The same age
- Love your favourite tick-tock
- Not as many brothers and sisters as you

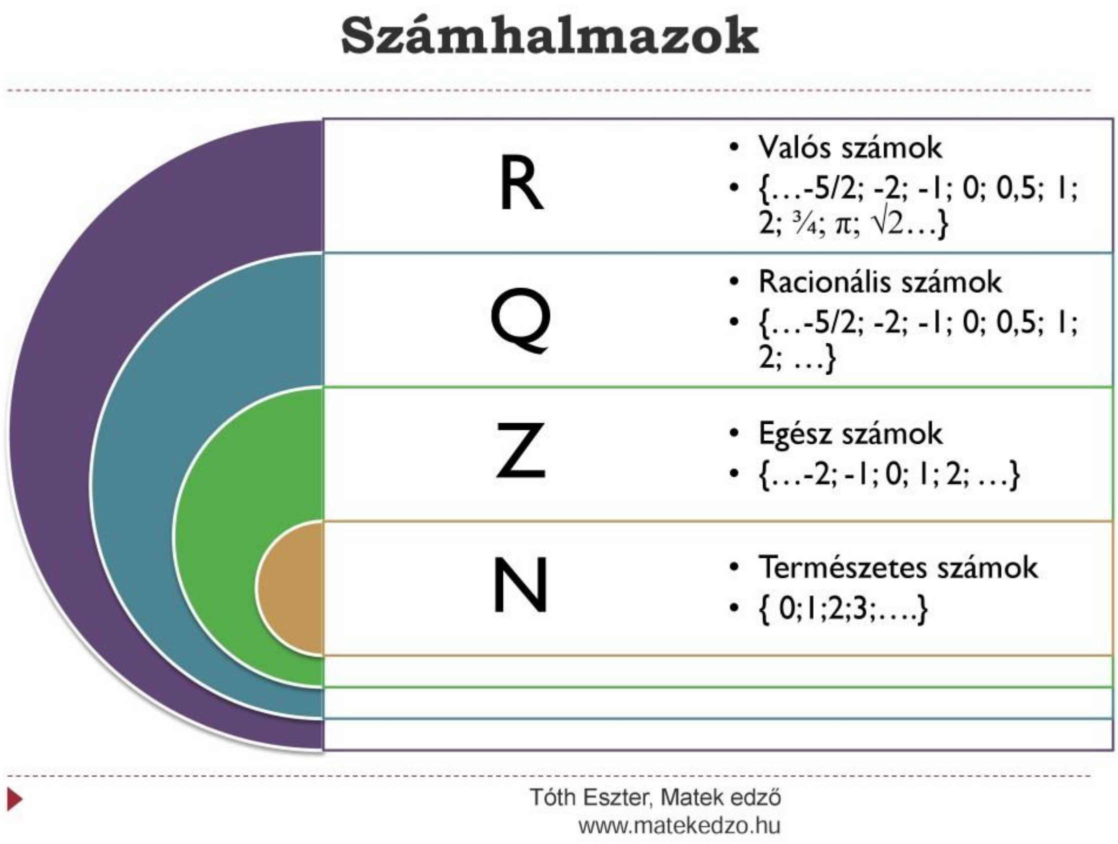
Write down this information about everyone in excel, → value urge

Get in a group with the same number of siblings
→ let the 1&2 brothers and sisters stand together

engraving	merger	difference

common elements of 2 sets	All elements of two sets	Only the elements of one set, without the elements of the other set
Setting a common example with children	Who likes fries, veggie burgers and cola?	Some people like hamburgers, some people like French fries and some people like kolas. Those who like one or the other (unio) go to a meki. Those who like all of them (metszet) go there for a menu.

Sets of numbers



Crumbs

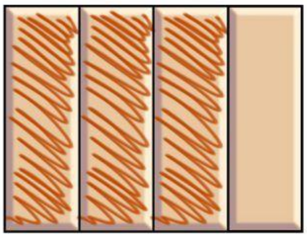
A fraction can express a number less than a whole number.
Think of a fraction as breaking a whole bar of chocolate into pieces of the same size.

The parts of the fraction. A fraction is divided into an upper and a lower part by a fraction line. The upper part is called the numerator, the lower part the denominator.
Denominator of the fraction
The denominator shows how many parts the whole is divided into.
Sticking with the chocolate example above, if the whole bar of chocolate is broken into 4 pieces, then the denominator is 4.
We say a quarter.
If we break it 6 ways, we say it's a sixth.
Counter of the fraction
The counter shows how many of the broken pieces have been bought.
Example 1 for the fraction

If the whole bar of chocolate is cut into 4 pieces, and 3 of these are eaten, the counter will count 3.

If the numerator is 3 and the denominator is 4 $\left(\frac{3}{4}\right)$, we say that there are three quarters. So we have eaten part of $\frac{3}{4}$.

The example above is illustrated with a drawing:


Three of the four parts are coloured.

The board is broken into 4 parts. We have eaten 3 of them, so 3 are coloured in.

So now we have coloured the $\frac{3}{4}$ part of the whole board.

Let's look at another example:

Example 2 for the fraction:

The task is to colour in the $\frac{2}{8}$ part of the rectangle:


Two of the eight parts are coloured.

The rectangle is divided into eight parts, so 2 of these must be coloured.

I brought in 30 granola bars, how many did you eat? → how many in a fraction?

Draw 9 cubes (we on the board they in the notebook)
- Synonyms for 3/6-oda
- The 1/2 and 1/3

Games2

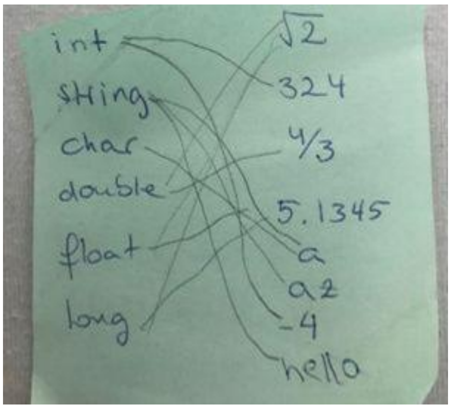
1. They pull you out of a hat and have to pair up

There will be INT, FLOAT, STRING type and there will be ARITMETIC OPERATIONS PL- With 18 children, we can play the following game with 1-1 slip of paper with the next 18 words on it. First they have to pair up, then the pairs have to find an arithmetic operation pair (e.g. INT - 5, INT - 9 4 children, find AM2 - subtraction two children) and write down and perform the operation six by six. Groups of 3x6 will emerge in this case. Obviously division/subtraction is not good for the String group,you have to pay attention to this. Figure it out for yourselves

Int - 5
Int - 9
Float - 0.33
Float - -0.66
String- nice good
String day
Arithmetic operation 1 - division
AM 2 - subtraction
AM 3 - addition

2. Connecting game on a paper base

Connect the numbers in the columns with the variable type. A number or a type can have more than one pair!
Example:



int:	324
	-4
double, float, long:	324
	4/3
	5.12345
	4
char:	a
String:	"hello"
	"az"

C# exercises2:

```
int egeszSzam = 2;

float lebegoPontosSzam = 0.33f;

double tortSzam = 3 / 4;

long nagySzam = 100_000_000_000_000_000L;

char karakter = 'c';

char szivKarakter = '♥';

String szoveg = "ez itt csupán karakterek sorozata:)";
```

3. Lesson: Structuring variables (Unit 3 material)

object - a storage/box/storage of logically related data/representation of reality that contains the object's properties

array - a group of karekter types (memory area of defined length, number of elements)
list - dynamically growing, list of anything
map - key + value
class/class café
object chair ,
list/map/block store the foot# color
object table ... etc
list/map/block stores the size(cm) of the shape

set - set

```
int[] egeszSzamoknakTomb = new int[2];
egeszSzamoknakTomb[0] = 1;
egeszSzamoknakTomb[1] = 2;

int[] egeszSzamok = { 1, 2, 3, 4 };

List<char> karakterekListaja = new List<char>();

List<char> karakterekListaja2 = new List<char>() { '♥', '♦', '♣', '♠' };

Dictionary<string, string> telefonKonyv = new Dictionary<string, string>();
telefonKonyv.Add("Pistike", "+36301111111");
telefonKonyv.Add("Juliska", "+36302222222");

HashSet<string> szinekHalmaza = new HashSet<string>();
szinekHalmaza.Add("Kek");
szinekHalmaza.Add("Piros");       //Mivel ez egy halmaz így csak egyszer
szinekHalmaza.Add("Kek");         //fogja a kék színt tartalmazni

Allat hogolyo = new Allat("Hogolyo", "Feher");
Allat foxi = new Allat("Foxi", "Fekete");

public class Allat
{

    string nev;
    string szin;

    public Allat(string nev, string szin)
    {
        this.nev = nev;
        this.szin = szin;
    }
}
```

Games3

4. CLASSES , OBJECTS

cabinet- folder- file
diary - tickets ; doctor's office,

pair finder by variable type and value (int double float, long, string, char, object (anything goes))

- *pulling from boxes, looking for each other*

group work : draw the sets on the board

- *mammal - dog + cat - labrador (colour:yellow height:40), huskey(grey big), dachshund(brown tiny) + Siamese(brown tiny), maine coon(grey big), naked(leather tiny)*
- *class - class + class - object (property , tul)*

make up characters (3,4 legged dog and arrange/learn them)

```csharp
Cica cirmi = new Cica(nev: "Cirmi", szin: "Cirmos");
Kutya picur = new Kutya(nev: "Picúr", szin: "Barna");
Allat kepzeletbeliAllat = new Allat("KepzeletbeliAllat", "KepzeletbeliSzin");
Console.WriteLine(cirmi);
Console.WriteLine(picur);
Console.WriteLine(kepzeletbeliAllat);
class Kutya : Allat
{
    string hangja = "VAU";
    public Kutya(string nev, string szin) : base(nev, szin) { }
    public override string? ToString() { return $"{nev} vagyok, {szin} színű és ilyen hangot adok ki: {hangja}"; }
}

class Cica : Allat
{
    string hangja = "MIAU";
    public Cica(string nev, string szin) : base(nev, szin) { }
    public override string? ToString() { return $"{nev} vagyok, {szin} színű és ilyen hangot adok ki: {hangja}"; }
}

public class Allat
{
    public string nev;
    public string szin;

    public Allat(string nev, string szin) {
        this.nev = nev;
        this.szin = szin;
    }
    public override string? ToString() { return $"{nev} vagyok, {szin} színű és nem tudok hangot kiadni :("; }
}
```

```
Cirmi vagyok, Cirmos színu és ilyen hangot adok ki: MIAU
Picúr vagyok, Barna színu és ilyen hangot adok ki: VAU
KepzeletbeliAllat vagyok, KepzeletbeliSzin színu és nem tudok hangot kiadni :(
```

C# apprenticeships3:

Logic

What is Logic?

Logic is the science of the laws of reasoning

- Mathematical logic is the branch of logic that applies mathematical methods to the study of formal logic.
- Subject of the statement calculus and predicate calculus

Elements of the statement calculus

- Basic concepts:
- The definition of a statement is a proposition that can be clearly determined to be true or false.
- Statements have a logical value: a statement can be true or false.

- A sentence is a set of symbols to which we attribute meaning.
- In a declarative sentence, something is stated about something, i.e. the subject and the predicate are present.
- A declarative sentence is called closed if it refers to well-defined things.
- An unambiguously true or false statement means that:
  (a) cannot be both true and false (principle of non-contradiction)
  (b) it cannot be true or false (third principle excluded)

Paradoxes

- Paradoxes are sentences that are both true and false - Find the truth value of the sentence in the next frame:

a) If it were true, then "I am lying now", so I am not telling the truth, so the sentence is false.

b) If it were false, then it is not true that "I am lying now", so I am telling the truth, i.e. the sentence is true.

"MOSTLY LIE"

So if someone says, "I'm lying now", he is both telling the truth and lying, i.e. the sentence has no truth value.

The previous example was a version of the Bertrand-Russel paradox.

- A paradox is a sentence that seems to contradict a point of view or a mathematical theorem.
- It can occur as a result of incorrect evidence or by starting from incorrect premises.
- In the previous example, we did not have a statement, but only a sentence, which has no logical value.

The barber paradox

There is a barber in a village who shaves all the villagers who do not shave themselves. Who shaves the barber?

- Since the barber is also a villager, he must be shaved by the barber.
- On the other hand, the barber only shaves people who do not shave themselves.
- So the barber is shaving and not shaving
- Given statements can be used to derive new statements by combining them with the words "and", "or", "not", "if... then..." and "then... and only then...".

With such interconnections, elementary statements become so-called compound statements, whose logical value depends only on the logical value of the component elementary statements.

Igazságtáblázat

A	*B*	$A \wedge B$
i	*i*	*i*
i	*h*	*h*
h	*i*	*h*
h	*h*	*h*

A	$\neg A$
i	*h*
h	*i*

A	*B*	$A \vee B$
i	*i*	*i*
i	*h*	*i*
h	*i*	*i*
h	*h*	*h*

A	*B*	$A \Rightarrow B$
i	*i*	*i*
i	*h*	*h*
h	*i*	*i*
h	*h*	*i*

Games

1. Say Logical sentences (try to draw a connection between the ideas) . like we did
2. AND OR NOT

Megjegyzés: A konjunkció ún. igazságtáblázata:

A	B	$A \wedge B$
1	1	1
1	0	0
0	1	0
0	0	0

Példa:

A =„A 2 természetes szám."
B =„2 nem nagyobb 1-nél."
$A \wedge B$ =„A 2 természetes szám *és* nem nagyobb 1-nél."

Az A állítás logikai értéke 1, a B állításé 0, így a $A \wedge B$ állítás logikai értéke 0, azaz az állítás hamis.

3. *Megjegyzés*: A diszjunkció igazságtáblázata:

A	B	$A \vee B$
1	1	1
1	0	1
0	1	1
0	0	0

Példa:

A =„A rombusz minden szöge 90°-os."
B =„A rombusz oldalai egyenlő hosszúságúak."
$A \vee B$ =„A rombusz minden szöge 90°-os *vagy* a rombusz oldalai egyenlő hosszúságúak."

Az A állítás logikai értéke 0, a B állításé 1, így a $A \vee B$ állítás logikai értéke 1, azaz az állítás igaz.

Megjegyzés: A $\neg A$ állítást A negáltjának nevezzük.
A negáció igazságtáblázata:

A	$\neg A$
1	0
0	1

Példa:

A =„A háromszög szögei egyenlők."
B =„Két páros szám összege mindig páros."
$\neg A$ =„A háromszög szögei *nem* egyenlők."
$\neg B$ =„Két páros szám összege *sohasem* páros."

(http://users.atw.hu/vizsgazo/mat_elmelet_elemei/9_logika.pdf)

4. Creating example sentences for the logic board - getting children to say the examples on the boards.

In C#, the boolean

```
bool jatekVege = false;

if (jatekVege)
{
    //pontok összevetése,
    //káryták összegyűjtése
    //visszavágó? if(visszavágó)
    //     /                  \
    //keverés, osztás      pakli elcsomagolása
}
else
{
    //lapot huzunk
    //jatekos 1 lerak 1 lapot
    //jatekos 2 lerak egy laokt
    //.....
}
```

Algorithms

In our daily lives, we encounter problems and tasks that we have to solve when we interpret the signals from our environment.

It can be a problem if you are hungry, cold, have homework to do, or dirty dishes to look at. These tasks and their solutions come naturally to us; we don't even have to think too hard about the series of steps we have to go through to solve them.

To solve the problem, to achieve the desired result, we carry out different actions, perform steps, evaluate conditions, on the basis of which we decide on our next action. By analysing the tasks, we determine the sequence of actions or activities that will require the least effort and lead to the desired result.

There are many ways to solve a problem. It is always a good idea to consider the possible solutions and sources of problems before taking action. This process is called planning.

For example, if we need to get to a point in the city, we plan our route. We consider which route is the shortest, whether there are any roadblocks, what means of transport to use, what the weather is like. We evaluate the problems encountered and make a decision.

When designing a series of steps, we should aim to find the best possible solution. The more steps in an activity, the longer the solution takes and the greater the potential for error.

The concept of an algorithm

The process of carrying out a series of defined steps to solve a problem is called algorithmisation.

In the most general sense, an algorithm is a plan. By thinking through a sequence of actions to be performed step by step, an algorithm is created to achieve a given goal. The steps are carried out one by one according to precise rules. Another way of putting it is: in an algorithm, we specify a solution to a problem that can be carried out in a finite number of steps. The ideas we have for solving the problem are transformed into instructions, the sum of which results in an algorithm.

Algorithmisation is the process of breaking down our activities into small elementary steps. For example, saying that we are hungry is not a sufficient solution to the problem of eating. To get to the point of eating, we need to think through several steps. Is there food in the house, or do we have to cook? Where can we find food, if there is food? How do we get there? If there are several types of food, what do we want to eat? Is there a pot to eat from? Where is the dish? In summary, many questions may arise. A possible algorithm to solve the problem is to go to the kitchen, open the fridge, take a plate from the cupboard, take the food out to the plate, heat it up if necessary, sit down on a chair at the table and eat.

Elements of the algorithm

The algorithms consist of different types of steps. First, we have to define the problem, the starting point and the goal, i.e. the end point. These are the steps that unify our algorithm.

For example: the problem is that the plate is dirty, the goal is to get it clean. In the design process, we define clearly executable generic operations that can be implemented given the previous step. For example: in order to get the plate clean, I open the tap, pour dishwashing liquid on the sponge, wash the plate.

After planning the actions to be taken, it is necessary to think about the potential problems and failures that may occur, and then to identify the possible actions to be taken if the problem is eliminated and if it is not. This type of action is called a condition. When examining a condition, all possible solutions should be considered and, if necessary, new steps should be introduced or a previous step should be reverted to.

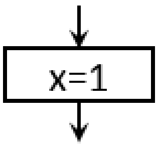
For example: whether the conditions are right for washing the dishes, whether there is water or dishwashing liquid. If there is everything, we can wash the dishes, if there is no dishwashing liquid, we have to go to the shop and buy it, and only then can we wash the dishes. Another condition may be whether the dish is clean when the washing up is done. If it is, we finish the dishwashing, if not, we scrub a little more.

Basic elements

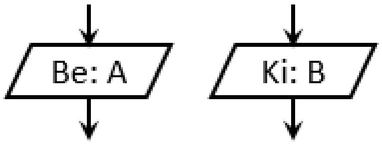
A flowchart is nothing more than a graphical representation of algorithms. Each elementary step is represented by shapes, which are then connected by arrows that define the order in which each action is performed. Different shapes are associated with different types of actions.

Start and End symbols: these ellipsoidal shapes define where to start the algorithm and when to end the sequence of steps. The maximum number of Start symbols in a flowchart is one. Only one outward arrow can start from it and no inward arrow can be connected to it. There can be more than one End symbol in a flowchart, but always exactly one end symbol is sufficient. One or more arrows may point in its direction, but none may exit from it.

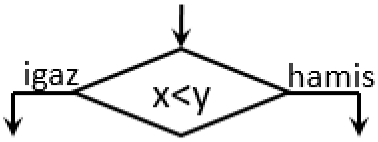
Elementary activity: a simple and straightforward step in a rectangular shape that does not require further explanation. For example, the action in the figure below is a value assignment, which means that the value of the quantity (also called variable) X should be 1 from now on.



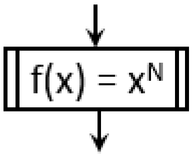
Input and Output symbol: most algorithms need some kind of input data. This will usually only be provided by the user during execution and this value will be used by the algorithm (similarly to the example showing generalisation). The resulting value is always stored in the named quantity in the formula. Other times the algorithm returns a result, it wants to communicate something to the user. The output could be, for example, a value of a quantity (as in the example below), or it could be a simple text, such as a "Hi!" message. Text constants are always in a cat circle, so we can distinguish them from quantity names. Only one of these symbols can have an arrow pointing outwards.



Condition symbol: the most complex element of flowcharts in terms of structure and function. A rhombus can have one statement per vertex. A statement is a logical expression that must be either true or false. Once we have evaluated this condition, i.e. determined its truth, we have two options. If the condition is true, we must continue execution along the arrow towards which the word true (possibly yes) is written. If the statement turns out to be false, we must continue along the arrow labeled false (possibly no). So there are always two labelled arrows pointing outwards from the side of the diamond.



Nesting symbol: sometimes in a flowchart you need to apply a sequence of instructions that is essentially a series of steps. If it has been defined and described before, but you don't want to do it again, you can use the double-sided rectangle as the embedding symbol. This implements one of the algorithm modification methods discussed earlier. In the figure below, we can see that we want to compute the power N of the quantity x, which is basically an activity involving a number of steps, but here we just refer to this algorithm, probably already described earlier, as a kind of new elementary step. The same can be said about the arrows associated with the embedding symbol as for the elementary activity.



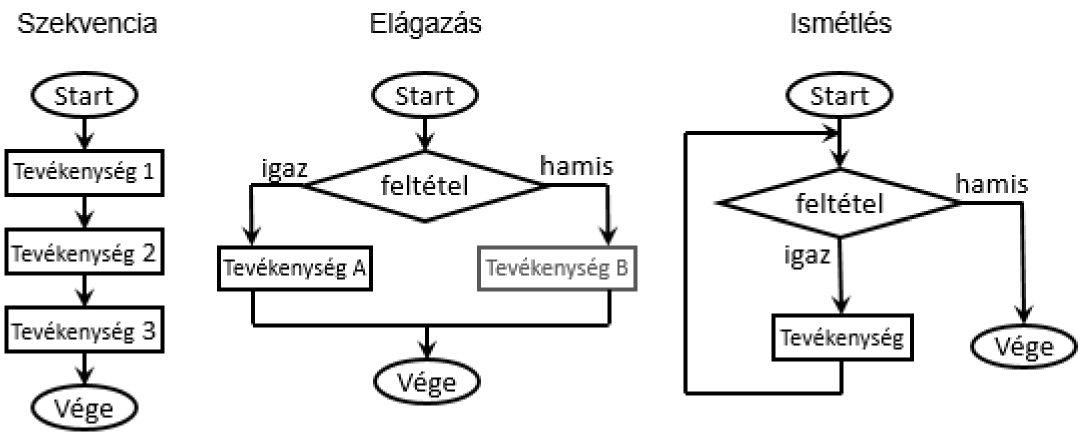
Some elements may therefore have more than one inward arrow associated with them. In such a case, we can use the notation where the arrows going towards the shape meet at a point and from there only one arrow continues towards the shape.

Basic structures of the algorithm for a flowchart

Of course, the three basic structures of the algorithms can also be described by a flowchart. Sequence here means activities linked by arrows. The arrows clearly define the order of execution, because by linking them together only a single path is created. From a shape, you can only continue along the arrow if the activity has been fully executed. It is not necessary to arrange the activities from top to bottom or left to right, since the arrows determine their relationship to each other anyway.

Branching can be achieved by a condition. Arrows leaving the ruin point to an activity or group of activities. However, once they are written, the two branches must merge, i.e. two arrows meet at a point and continue to move in a single line. The interpretation is as follows. When you reach a condition, you must evaluate the logical expression in the diamond. If this condition is met, proceed to the `true branch` and execute the instructions there and after reaching the point where the arrows meet, continue with the next activity after the branch! Conversely, if the condition is not fulfilled, continue along the `false branch`, and finally, in this case, you will also be directed to the instruction after the fork. So exactly one of the two branches is executed (depending on the condition). In either case, the execution continues in the same way. Sometimes a special case of a two-way branching is required, where one branch does not contain any activity. This means that either a step is executed or it is skipped. For future reference, it is advisable to place the activity to be skipped in the `true branch`, leaving the `false branch` empty (a single arrow to the meeting point).

The repetitions can also be described by a condition. In this case, one of the sequences of activities starting from the diamond containing the condition is such that, sooner or later, along the arrows representing the sequence, we return to the condition already covered. A loop is thus formed. There is no such loop on the other side of the condition. So, from the point of view of implementation, the condition is evaluated first. Depending on that, we either enter the loop, execute the instructions there, and then evaluate the condition again, or simply continue in the other direction. For the moment, we accept without explanation that the loop core, i.e. the instructions to be repeated, are always on the `true` side of the condition. So, in short, we can say that the iteration is done as long as the condition is true and as soon as it turns out to be false we move on. It is important to note that the instructions to be repeated must have an effect on the condition, because if this is not true, the condition will always remain true, i.e. we never exit the loop. However, an algorithm must be finite.



In the figure above, the three basic structures are illustrated in a flowchart. For the branching, `Activity B is` shown in grey because, as discussed, this branch can be empty, i.e. the activity can be skipped. As can be seen, branching and repeating both contain a condition. However, it is easy to distinguish them if we consider that in the case of branching, the two branches of the execution always meet later, while in the case of repetition this does not happen, but one branch always returns to the condition.

Games

1. Posted on

What kind of IT professional should I be?
https://progmasters.hu/wp-content/uploads/2020/05/PROGMASTERS-IT-SZAKMA-KV%C3%8DZ-1600x2000_feher.png

2. Posted on

Give examples of each basic structure

3. Task

How could the pancake making process be used to illustrate this?
Baking pancakes:

Palacsintatészta összekeverése csomómentesen

Olaj melegítése serpenyőben

Tészta serpenyőbe öntése

Tészta megfordítása

Ha mindkét oldala megsült a palacsintának Akkor

Palacsinta kiborítása tányérra

Egyébként

Palacsinta megfordítása

Palacsinta kiborítása tányérra

```
const string turospalacsinta = "turospalacsinta";
const string csokispalacsinta = "csokispalacsinta";
const string lekvarospalacsinta = "lekvarospalacsinta";
Console.WriteLine(palacsintaKeszites(lekvarospalacsinta));

Palacsinta palacsintaKeszites(string milyenPalacsinta)
{
    Palacsinta uresPalacsinta = Palacsinta.keszites();

    return feltetRarakas(uresPalacsinta, milyenPalacsinta);
}

Palacsinta feltetRarakas(Palacsinta palacsinta, string milyenPalacsinta)
{
    if (milyenPalacsinta.Equals(turospalacsinta)) { palacsinta.feltet = "turo"; }
    else if (milyenPalacsinta.Equals(csokispalacsinta)) { palacsinta.feltet = "csoki"; }
    else if (milyenPalacsinta.Equals(lekvarospalacsinta)) { palacsinta.feltet = "lekvar"; }
    return palacsinta;
}

class Palacsinta
{
    public string feltet { get; set; }
    private Palacsinta() { }
    //elkészítjük a palacsintát: bekavarjuk a tésztát,
    //kisütjük (milyen lépéseket hagytam ki? :) )
    public static Palacsinta keszites() { return new Palacsinta(); }
    public override string? ToString() { return $"Egy palacsinta vagyok ilyen feltéttel: {feltet}"; }
}
```

```
//készíts 10 palacsintát
for (int i = 0; i < 10; i++)
{
    palacsintaKeszites(csokispalacsinta);
}
```

Hatchery

The condition of generality

A property of an algorithm is generality. For a sequence of operations, it is important that it is not only useful once, but that it gives good results when applied to similar cases. An algorithm may have input and output data. The input data must be given at the beginning of the algorithm. This is used, an operation is performed on it, and the result is obtained as output data at the end of the algorithm. In such cases, a general formulation of the algorithm is necessary.

For example, if you're designing an algorithm to add two numbers together, you need to build it so that it can add any numbers with similar properties. In this case, even if we change the data, the process will still give good results.

Another example is the sequence of steps to search the dictionary. The steps of the algorithm written for this purpose must be formulated in a general way so that it can be used to look up any word in the dictionary, not just a specific term.

The condition of finiteness

The algorithm is characterized by finiteness. An algorithm consists of a sequence of steps. It is important that the result is obtained within a reasonable time after each step. The algorithm must be constructed in such a way that it finishes or gives a result after a finite number of steps.

An algorithm with infinite steps can be generated if the endpoint is not precisely defined, or if infinite iterations are used.

You may need to repeat a step several times to solve a problem. Always structure the repetition by specifying the number of repetitions in advance, or by including an expected condition in the repetition that, if met, will stop the repetition.

For example, a repeated step in the dishwashing algorithm is to clean the dish with a sponge with dishwashing liquid. In the algorithm for this activity, a condition must be built in to stop the repetition: wipe the dish with the sponge. When it is clean, stop washing. If not, go back to cleaning, then check the condition again.

The condition of succession

The algorithm is characterised by the property of sequentiality. In the algorithm, each step must follow the others in a specific order. Each step must be followed by another step, except the last one.

Each step must be executable, and the sequence of steps must be defined in such a way that any step executed is clearly followed by the next one.

When breaking down a process into steps, you cannot leave open, undecided conditions. You cannot leave the evaluation of a condition or the determination of the next step to the executor, this must be specified in the algorithm.

For example, in an algorithm for closing a door, the order of some steps can be reversed in the design phase. The steps are: go out the door, insert the key in the lock, close the door, turn the key. The two middle steps are interchangeable. It makes no difference to the achievement of the objective which is performed first. However, this has to be decided before the algorithm is created. It is not possible to include a step in the algorithm that either closes the door or puts the key in the lock after the exit. The steps must follow a specific sequence.

In summary:

In computer science, an algorithm usually consists of the following elements:

- determining input data (input)
- execution of a finite number of steps (program) of selected operations or instructions (e.g. cycle, select) that control the execution sequence
- execution of qsteps one after the other (sequence)
    - Start; beginning
    - N←10; how often
    - I←1; where we are
    - (loop execution)
    - Stop; finiteness
- repeated execution of specific steps (cycle)
- repeat until i≤n is satisfied
- otherwise (i.e. if i≤n is not satisfied)
- exit the loop
- determination of output data, results (output)

Algorithms and Logic (C#)

True false in algorithms
Programming this in C#

```
bool jatekVege = false;

if (jatekVege)
{
    //pontok összevetése,
    //káryták összegyűjtése
    //visszavágó? if(visszavágó)
    //     /                  \
    //keverés, osztás      pakli elcsomagolása
}
else
{
    //lapot huzunk
    //jatekos 1 lerak 1 lapot
    //jatekos 2 lerak egy laokt
    //.....
}
```

Cycles, Statements

Cycle		
For	Condition: how many times IMPORTANT: it's all about the numbers!	Write "X" 15 times
While	As long as the condition is not false (runs as long as it is true)	Walk until it's light
Do While	and others.this 1x will run anyway, even if the condition is false	Walk until it's light, but you'll still walk when it's dark
Branching		
If else		If your friend likes animals, you can ask: • If you like dogs, should I buy you a dog toy? • If you like cats, should I buy you a cat toy? If your friend says they like dogs, you can buy them a fun dog toy.
Switch case	There is a fixed set of options (set of values), each output of which is worth defining	Or you can ask: • Do you prefer things with dogs, cats or other animals?

EBranch

Game

1. IF ELSE:

for(string item in items){
clap the item
}

2. SWITCH CASE:

```
for(string item in items){
clap the item
}
```

Theory

- Game processing to see what it's good for
- Algorithm flowchart to plot the game we played
- What exactly is it good for? When do we use it?

Use of switch-case and if-else control structures:

-Both control structures are used when you want to put a so-called branch in your code, because the run can go in several directions. For example, if we are talking about a game, our program will run until someone wins, then we stop it. Here, the branching part is about checking if someone has already won? If yes it stops if not then the program continues to run, so that's 2 possibilities.

-A practical use is to use the switch-case structure for a predefined number of branches (e.g. what day is it? There will always be 7 days) and the if-else structure for all other cases.

//TODO -put code also!!!! MINDENHOVA!

C#

FOR

Game

1. FOR:

```
for(int i = 0; i<n ;i++ ){
gugolj i-times
}
```

2. FOR EACH:

```
for(string item in items){
clap the item
}
```

Theory

- Playing to see what it's good for
- Algorithm flowchart to plot the game we played
- What exactly is it good for? When do we use it?
- Used when you want to walk through all the elements of a data structure

C#

```
int[] szamok = { 1, 2, 3, 4, 5 };

foreach (int szam in szamok) {
    Console.WriteLine(szam);
}
```

WHILE - DO WHILE

Game

1. WHILE

If Red Riding Hood and Martika wear a hairband, hop on one foot
- Béla: knows condonation (wears a hairband)
- Piroska & Mártika have a hair gum (they don't know anything)
- Abel: evaluating the conditioner ▯🖐☝✌☪ v HAMIS
- Anabell: will tell you if it should be implemented
- Orsi: knows what to do (jump on one foot) and tells you
- Others: task performer
- Karcsi: programmer (rewrites the code so they don't jump)

while (Red Riding Hood and Martika are wearing hair elastics) {
hopping on one foot
}

while (until someone runs out of cards) {
let the game go
}

2. (DO) WHILE:

do while (while Lacika has a high score){
Google down
}

Theory

- Playing to see what it's good for
- Algorithm flowchart to plot the game we played
- What exactly is it good for? When do we use it?
- It is very rarely used because it can be dangerous, it is not really seen in a corporate environment nowadays

C#

Probability calculation

What is random?

It is important that children find random things in life for themselves. Will they find it?

Accidental is that which we would not have expected to happen, and *accidental is that which* has occurred seemingly without cause, irregularly, unforeseeably, meaninglessly or "without reason". There are *accidents* - for there are times when we would not have expected or anticipated certain things - and there are, according to many, no *accidents*, only regularity, i.e. *the law of cause and effect* (in other words, 'karma').

Mathematics and mathematical statistics also tend to conflate the two terms, but the scientific terms are now clear and familiar, and have long been accepted as unambiguous in professional language.

The English term "random" clearly means a *random occurrence*, but in Hungarian it is often translated *as chance.* This is why, when the Hungarian language speaks of *chance*, it increasingly adds the phrase "random" to indicate that it is in fact referring to the less commonly used term "*chance".*

Random = random

Randomness in nature

https://sokszinuvidek.24.hu/mozaik/2016/10/07/eszrevettetek-kaprazatos-geometriai-mintazatokkal-ajandekoz-a-termeszet/

Man seeks order in nature, patterns, distributions, regularities. To support these, he collects information and builds up systems.

We often consider an event to be random - or more accurately, *chance* - because we are unable to gather enough information to predict the outcome of the event. Other times, it is because its occurrence, its happening, seems incomprehensible, meaningless, useless to human reason.

In other cases, the outcome of the phenomenon is unpredictable in principle, and therefore inherently random. An example of such an event is the decay of radioactive nuclei. The probability of these events can be described and analysed, but the specific occurrence of an event is unpredictable.

In practice, these two phenomena are confused, since it is practically irrelevant whether the outcome of a coin toss (heads or tails) could be predicted in principle with perfect knowledge of the physical properties of the coin and the air currents (mass, velocity, friction, etc.).

Reflections on chance and mathematics : https://mek.oszk.hu/05000/05028/html/dialogmatek0003.html

So if we flip a coin? Which side will it fall on? Can we predict?

Coin toss game:

1. Guess what we're going to throw
2. Let's write down
3. Flip the coin
4. Write down which side you fell on
5. Repeat 50x

Write this down in a process diagram

Which cycle can be assigned?

Do the game (everyone gets a coin and a piece of paper)

Comparison at the end

- Is there a pattern to how the coin fell?
- Is there a pattern in the results?
- How often did it fall this way and that?
- What determines which side it falls on? What influences the coin?

Let's keep a record of how many times someone has been hit on the head or on the head
How much is enough information?
Can we predict which side it will fall on next?

Examples

1. What is the likelihood of it falling on your head? On writing?
2. What is the probability of a roll of the dice being a 1?
3. If we have a hat with 3 white and 6 black paper galaxies, what is the probability of drawing white? And black?
4. What is the probability that we roll more than 4?

Combinatorics

Theory

Combinatorics is the branch of mathematics that deals with finite sets and the key question is: how many ways?
Permutation is when we arrange all things in order. Example.

A combination is a situation where we select things in such a way that the **order of selection does not matter.** In a combination, we know exactly how many items we have, and we have to select a given number of items (given by the task) from these items, so that the order of selection is irrelevant. (So it doesn't matter where we put the items or people, because we don't know their exact location.)

And **variation is** when you select and arrange some things in order, so **their order matters**. For example, there are 10 children in a running race, but 3 podium places count.

Games

Permutation:
Make groups of 5 - how many ways can you arrange yourselves?
How many ways can you line up if Anna and Béla want to stand next to each other?

Combination:
There are 5 people waiting for the lift, but the lift is only for 3 people. How many different ways can you take the lift on the first round? (order does NOT matter)

Variation:
There are 5 people waiting for the lift, but the lift is only for 3 people. How many different ways can you take the lift on the first round? (The order of who enters first, second and third is important)

Examples

You can play these with them

1. Get into groups of 5 and guess how many ways you can get into the lift
2. Get into groups of different sizes and guess how many handshakes it takes to get everyone to shake hands
3. Stand in a circle and guess how many different ways you can stand up.
4. ...etc.

1. Egy virágárusnál 7 különböző fajta virágból lehet választani. Hányféleképpen vehetünk
   a. 5 szál különböző virágot?
   b. 5 szál virágot?
   c. 10 szál virágot?
2. Egy 20-fős tankör tagjai megajándékozzák egymást olyan módon, hogy beteszik a neveiket egy kalapba, és mindenki kihúz egy nevet, akinek ajándékot kell vennie. Mi a vsz.-e, hogy lesz olyan ember, aki a saját nevét húzza ki?
3. Egy 20-fős tankörben mi a vsz.-e, hogy legalább két ember egy napon ünnepli a születésnapját?
5. Egy 20-fős tankörben hányszor csendül a pohár, ha mindenki mindenkivel koccint egyet?

Gy2.Four people go to school together. In how many different orders can they enter the school?

Gy3.A group of ten people is seated at a round table. In how many different orders can they sit down?

Gy4.A group of ten people stand in a circle. How many ways can the order of the company be?

Gy7.A group of six people is standing in front of the four-person lift. How many ways can we choose the first 4 passengers?

1. Out of nine different colours, how many 3-colour flags can be made if no colour can be used more than once?

2.How many 3-colour flags can be made from nine different colours?

3. Roll a die four times in a row. How many different series of rolls can you get?

4. There are 5 white balls and 5 red balls in a box. What is the probability of drawing a white ball first and a red ball second? → Point them to the theory of recoil that makes a difference

16. 3 dice are rolled simultaneously. What is the probability that the point value on top of all three dice is greater than 4?

Unity links

Unity introduction/installation/get to know

https://www.youtube.com/watch?v=sKUxsHUqcR8&ab_channel=Andr%C3%A1sMihalik

Developing a coin

https://www.youtube.com/watch?v=UPBtFI_NERo&ab_channel=Kermitd77

Show cube

https://www.youtube.com/watch?v=ouGJJzNPsSk&ab_channel=AlexanderZotov

PROG HALADO

What is data, how exactly is it built up? What are the types (small repetition)
What are the classes? How are they structured? How do we recognise them, what are they for? How-where do we use them?
More complex data structures (e.g. trees, graphs) More for interest than to teach everything but important to understand when is it ideal to use them and why?
Repetition of the above
Unity → coin flip integration in unity
Creating rules, demoting, evaluating, modifying if necessary Talking about cyclical thinking - a bit "Frame numbers" how many cards How many rounds How many castes How many aspects How many points
Castes & facets design help with EE: 4. comfort zone - superpower from there? 5. Which side are you on? Which set are you on? 6. Processing
Cyclicality rules, creation, repetition
Cyclicality rules, creation, repetition

Game Design/Application Development

Prototype rules

30 pack of cards
100 points between variables
the player decides which value to attack with
caste selection (select castes child)
scoring of inter-caste variables

- 4 - 6 variable children guess

coin toss to decide
draw 5 to 5 cards
winner hiv tells you what variable (nyerts = first)
the machine selects one of the three strongest at random
comparison
receive the points
simultaneously pulls the machine player
in the next round the second will draw a card
at the very end you can pop-up the button to start again

PROTOTYPE in annex

PROTOTYPE PROGRAMMING: Required Define classes in C#, represent them in unity
Definition of departmental responsibilities
Implementing the functions of the departments
Connecting the game loop, adding cards to the program
Depositing a card
Swipe card
Animating a deck that is about to expire

Decision Tree / Game Theory

Algorithm flowchart

Bibliography

Alfred V. Aho - Jeffrey D. Ullman: Foundation of Computer Science, 1992, Computer Science Press, New York

Brookshear, J. Glenn: Computer science: an overview 9th ed. (International ed.), cop. 2007, Boston, MA: Pearson; Addison Wesley

Cormen, Leiserson, Rivest, Stein: Introduction to Algorithms, 1990, MIT Press and McGraw-Hill.

Péter Csala - Arthur Csetényi - Béla Tarlós:Computer Science Basics, 2001, ComputerBooks, Budapest

Efraim Turban - R. Kelly Rainer, Jr. -Richard E. Potter: Introduction to Information Technology, 2001, Wiley

Zoltán Kátai, C language and programming, 2008, University of Debrecen, Debrecen

[H1] *W. Trede-A.Herkelmann-E-Schwarzer: Computer Fundamentals*, 1975, Műszaki Könyvkiadó, Budapest