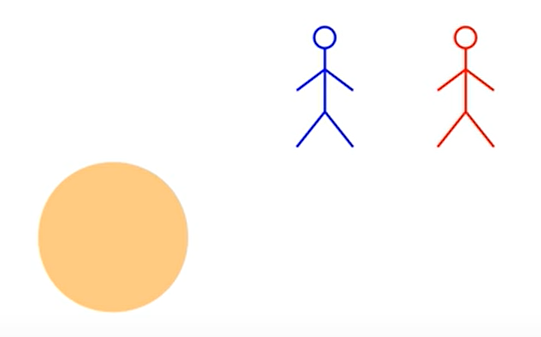
Assignment Cut and split

In this assignment we will develop a Software for, **fairly cutting process** for 2 -> n persons.

**How to split a cake fairly between 2 persons?**



2 persons! how can we make a cut that will be fairly for each of them?

Situation

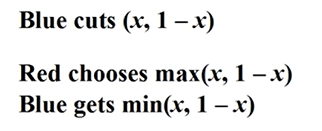
* Person 1 cuts and choose.  
  for his purpose (person 1) he won’t make the cut equal so he could get the largest size of the cake.

Fair algorithm for 2 Persons:

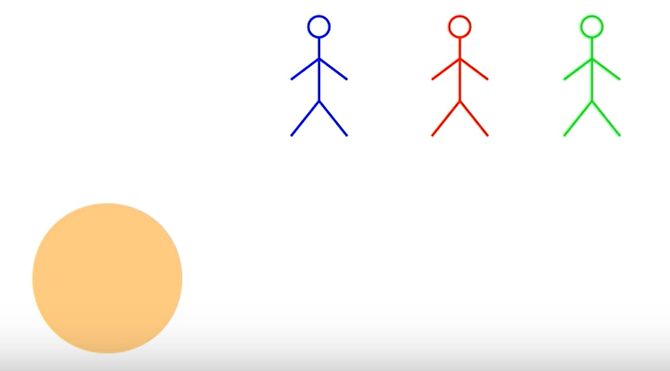
* Person 1 cuts the cake the way he wishes.
* Person 2 will inspect the slices and choose witch one he wants.
* The last slice will go to person 1.

**This algorithm will solve the fairly slice between 2 persons.**  
Explanation: person 1 cuts the cake, he will make the cut as even he could.  
**why?** if he doesn’t in all the cases he will lost and get the smallest portion of the cake, because the person 2 will choose the largest slice.

Algorithm conclusion:



**How to split a cake fairly between 3 persons?**



3 persons! how can we make a cut that will be fairly for each of them?

Fair algorithm for 3 Persons:

* Person 1 cuts the cake the way he wishes.
* Person 2 will inspect the slices and choose if he wants to cut it more.
* The last persons that cuts the slice will get it.

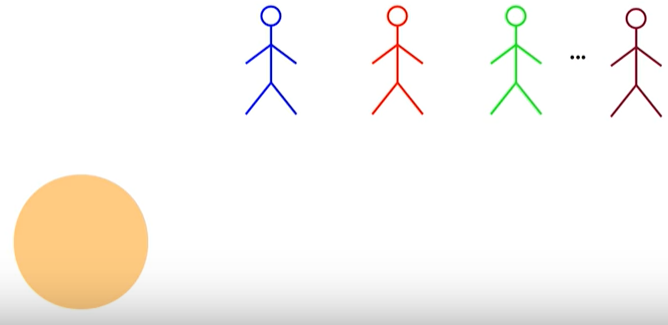
This algorithm will solve the fairly slice between 3 persons.  
Explanation: person 1 and 2 cuts the cake, they will make the cut as even they could 1/3.  
**why?** if they don’t in all the cases the 3 person will not cut and get the largest portion of the cake.

***Note that if someone cuts the cake to small than a 1/3, he will get that slice.***

When a person has received the slice. We are left with two pieces of the cake and 2 persons.

To solve this situation, we do it with the algorithm for 2 persons.

**Now that we have a solution for 2 and 3 persons, we can create an algorithm for N persons.**



**Algorithm for N persons.**

* Person 1 cuts the slice
* Person 2, 3…N have a choice to cut off some of the slices, the last person that cut gets slice.
* First slice gets 1/N
* Repeat this algorithm for N-1 persons.
* For final 2 persons remaining will do the cut and choose algorithm.

**Coding time**

Now that we have understand the algorithm’s how they work we must write the software for it.

Agents

An abstract class that describes a participant in a cake-cutting algorithm.

Agent(); Constructor for initialing name of agent

Agent(string); function that returns name of agent

**Abstract methodes:**

virtual float cakeValue() returns value of the cake for agent

virtual float cakeLength() returns the length of the total cake that the agent wants

virtual float eval(float,float) returns the value of the starting and ending point of the slice agent wants

mark(self, start:float, targetValue:float):

return "end" such that the value of the interval [start,end] is targetValue.

virtual float mark(float, float) returns the sum value of multiple slices.

virtual string print() print function

**Non Abstract methodes:**

string getName() return name of agent

string toString(vector<float>)

string toString(vector<vector<float>>)

float pieceValue(vector<float>)

vector<float> partitionValues(vector<float>)

Allocation

Private :

vector<Agent\*> agents;

vector<vector<float>> pieces;

public:

Allocation()

Allocation(vector<Agent\*> agents)

~Allocation()

string toString(vector<float> vec)

cut\_and\_choose

Implementation of the last diminisher protocol for fair cake-cutting among 2 agents.

Allocation asymmetric\_protocol(vector<Agent\*> agents) one cuts and the other chooses.

a list that must contain exactly 2 Agent objects.

Allocation symmetric\_protocol(vector<Agent\*> agents) both agents cut, the manager chooses who gets what.

a list that must contain exactly 2 Agent objects.

last\_diminisher

Implementation of the last diminisher protocol for fair cake-cutting among n agents.

LastDiminisher(vector<Agent\*> agents) list of Agent objects.

return: a proportional cake-allocation.

void LastDiminisherRecursive (float start, vector<Agent\*> agents, vector<int> activeAgents, Allocation allocation)

A recursive subroutine for last-diminisher.

start: the leftmost end of the cake that should be allocated.

agents: the list of all n agents in the original protocol.

active\_agents: list of indices of those agents who are still active (not allocated yet).

allocation: the current allocation (will be updated during the run).

return: nothing - the allocation is modified in place.