National Research University Higher School of Economics
Faculty of Computer Science
Bachelor's Program "HSE University and University of London Double Degree Program in Data Science and Business Analytics"

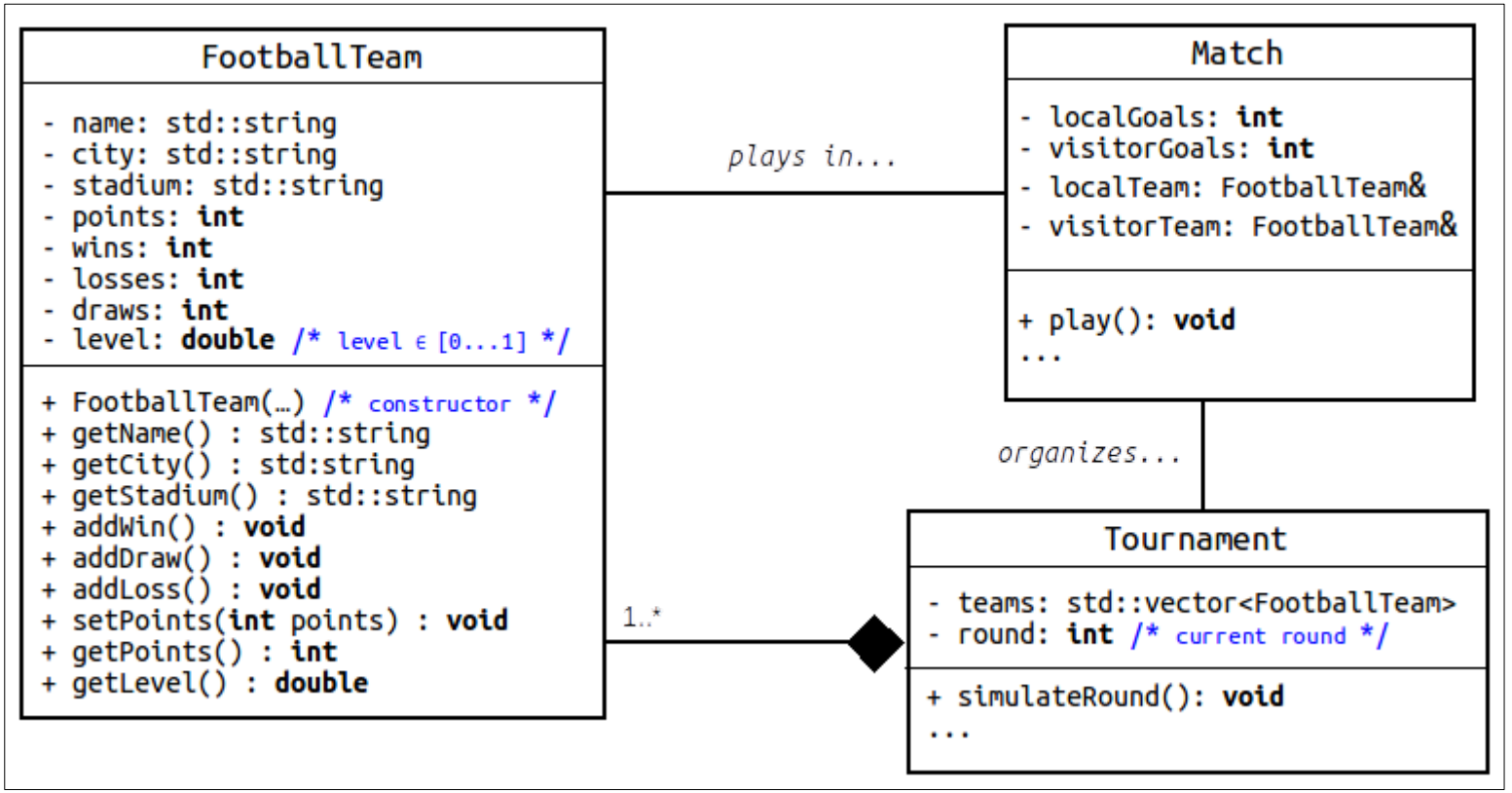
Introduction to Programming

Workshop #18

Fri 12.03.2021

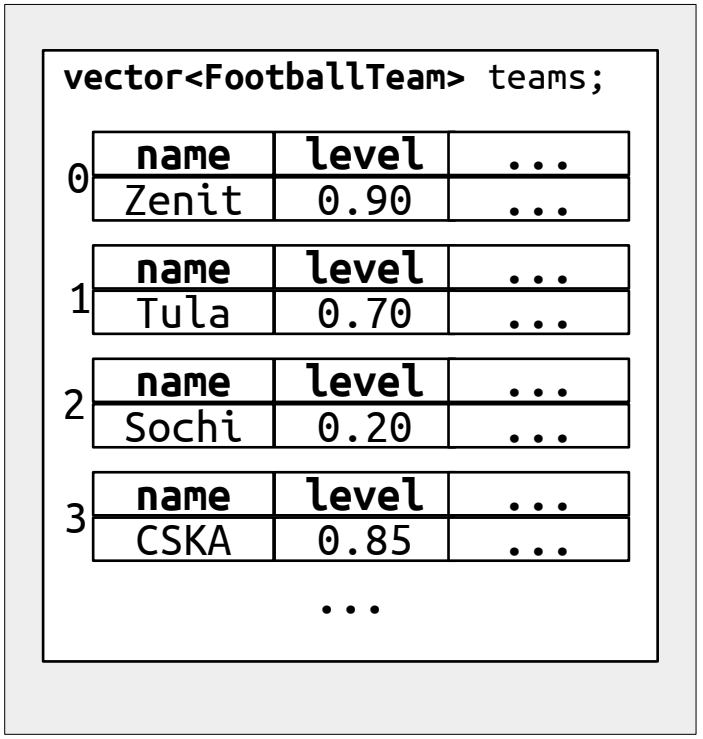
Julio Carrasquel

Our program – a football tournament...



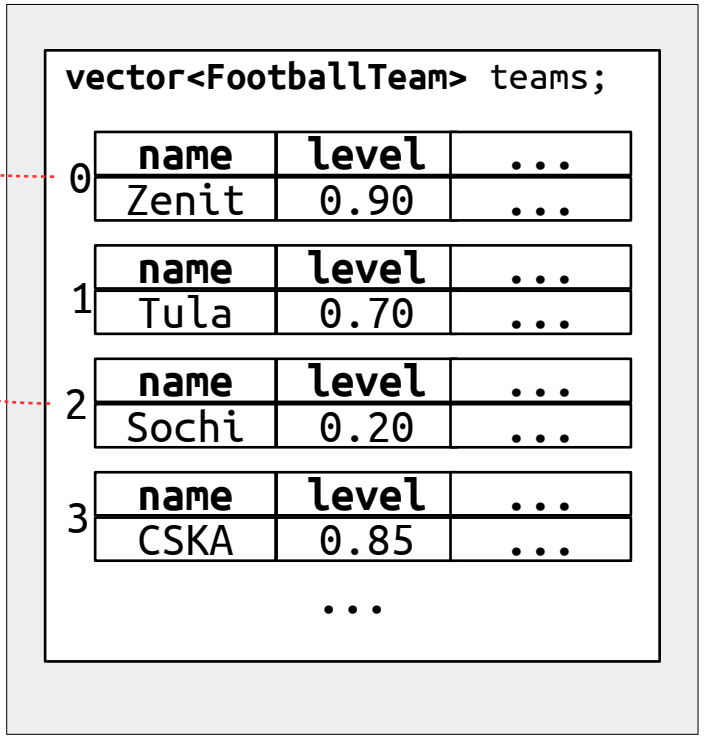
Simulating a match...

```
FootballTeam& localTeam = t.getTeams()[0];

FootballTeam& visitorTeam = t.getTeams()[2];
```



Simulating a match...

```
FootballTeam& localTeam = t.getTeams()[0];

FootballTeam& visitorTeam = t.getTeams()[2];
```



Reference objects (&)

Everything we'll do with "localTeam" and "visitorTeam" will modify them in the vector!

Simulating a match...

FootballTeam& localTeam =

name	level	...
Zenit	0.90	...

FootballTeam& visitorTeam =

name	level	...
Sochi	0.20	...

```
Match m(localTeam, visitorTeam);
m.play(); //simulate a match between the local team and the visitor team.
```

Simulating a match...

FootballTeam& localTeam =

name	level	...
Zenit	0.90	...

FootballTeam& visitorTeam =

name	level	...
Sochi	0.20	...


```
Match m(localTeam, visitorTeam);
m.play(); //simulate a match between the local team and the visitor team.
```

Simulating a match...

FootballTeam& localTeam =

name	level	...
Zenit	0.90	...

FootballTeam& visitorTeam =

name	level	...
Sochi	0.20	...


```
Match m(localTeam, visitorTeam);
m.play(); //simulate a match between the local team and the visitor team.


// how different teams are?
double diff = _localTeam.getLevel() - _visitorTeam.getLevel();
```

Simulating a match...

FootballTeam& localTeam =

name	level	...
Zenit	0.90	...

FootballTeam& visitorTeam =

name	level	...
Sochi	0.20	...


```
Match m(localTeam, visitorTeam);
m.play(); //simulate a match between the local team and the visitor team.


// how different teams are?
double diff = _localTeam.getLevel() - _visitorTeam.getLevel();
```

diff = 0.70

Simulating a match...

FootballTeam& localTeam =

name	level	...
Zenit	0.90	...

FootballTeam& visitorTeam =

name	level	...
Sochi	0.20	...


```
Match m(localTeam, visitorTeam);
m.play(); //simulate a match between the local team and the visitor team.


// how different teams are?
double diff = _localTeam.getLevel() - _visitorTeam.getLevel();
// maximum number of goals?
int maxGoals = ceil( abs(diff) * 10);
```

diff = 0.70

Simulating a match...

FootballTeam& localTeam =

name	level	...
Zenit	0.90	...

FootballTeam& visitorTeam =

name	level	...
Sochi	0.20	...


```
Match m(localTeam, visitorTeam);
m.play(); //simulate a match between the local team and the visitor team.


// how different teams are?
double diff = _localTeam.getLevel() - _visitorTeam.getLevel();
```

diff = 0.70

```
// maximum number of goals?
int maxGoals = ceil( abs(diff) * 10);
```

maxGoals = 7

Simulating a match...

FootballTeam& localTeam =

name	level	...
Zenit	0.90	...

FootballTeam& visitorTeam =

name	level	...
Sochi	0.20	...


```
Match m(localTeam, visitorTeam);
m.play(); //simulate a match between the local team and the visitor team.


// how different teams are?
double diff = _localTeam.getLevel() - _visitorTeam.getLevel();
```

diff = 0.70

```
// maximum number of goals?
int maxGoals = ceil( abs(diff) * 10);
```

maxGoals = 7

```
// how many goals in this match?
int numberOfGoals = uniform random number between 0 and maxGoals;
```

Simulating a match...

FootballTeam& localTeam =

name	level	...
Zenit	0.90	...

FootballTeam& visitorTeam =

name	level	...
Sochi	0.20	...


```
Match m(localTeam, visitorTeam);
m.play(); //simulate a match between the local team and the visitor team.


// how different teams are?
double diff = _localTeam.getLevel() - _visitorTeam.getLevel();
```
diff = 0.70

```
// maximum number of goals?
int maxGoals = ceil( abs(diff) * 10);
```
maxGoals = 7

```
// how many goals in this match?
int numberOfGoals = uniform random number between 0 and maxGoals;
```
numberOfGoals = 3

Simulating a match...

FootballTeam& localTeam =

name	level	...
Zenit	0.90	...

FootballTeam& visitorTeam =

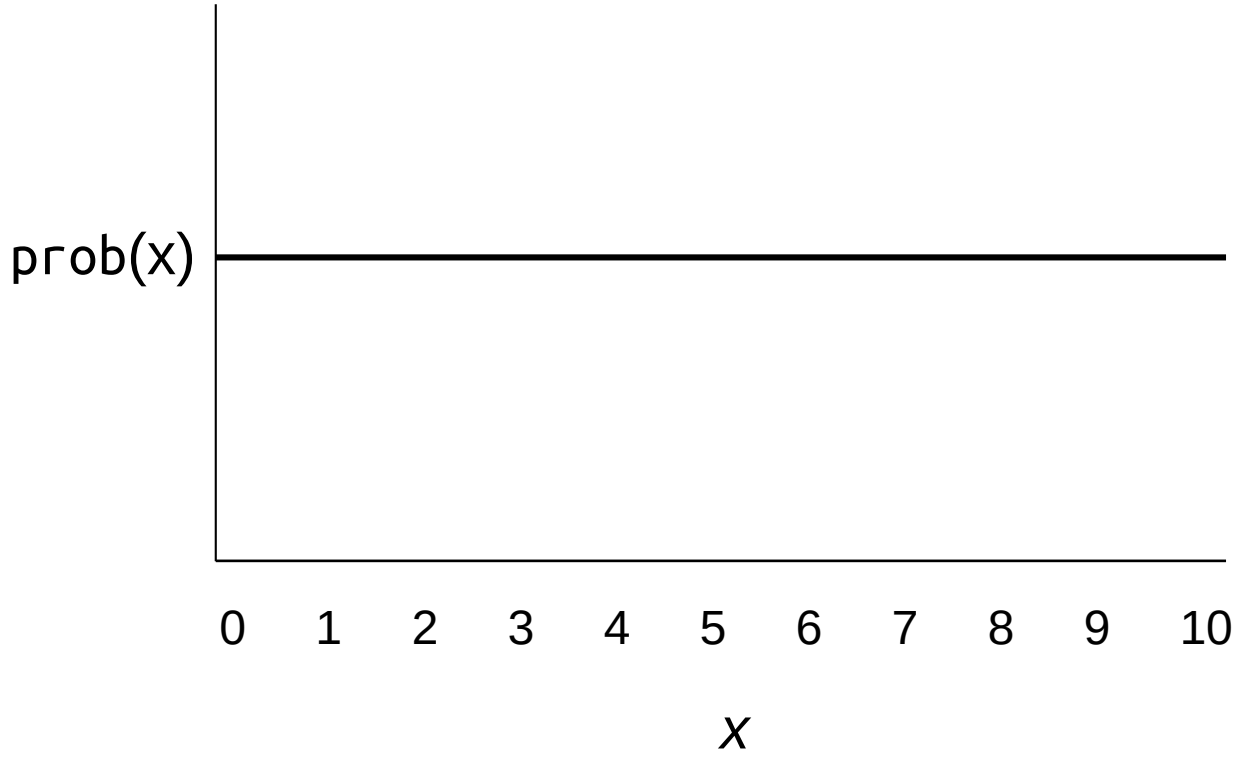
name	level	...
Sochi	0.20	...


```
Match m(localTeam, visitorTeam);
m.play(); //simulate a match between the local team and the visitor team.


// how different teams are?
double diff = _localTeam.getLevel() - _visitorTeam.getLevel();   diff = 0.70


// maximum number of goals?
int maxGoals = ceil( abs(diff) * 10);   maxGoals = 7


// how many goals in this match?
int numberOfGoals = uniform random number between 0 and maxGoals;   numberOfGoals = 3


// who scored the goals?
for(int i = 0; i < numberOfGoals; i++)
{
    double x = triangular random number (-localTeam.getLevel(), 0, _visitorTeam.getLevel());
                                                    (-0.90, 0, +0.20)
    if(x < 0) the local team scored a goal!
    else the visitor scored a goal!
}
```

About random numbers...

We have worked so far with **uniform** random numbers

```
prob(x): probability of generating x
x: random number
```

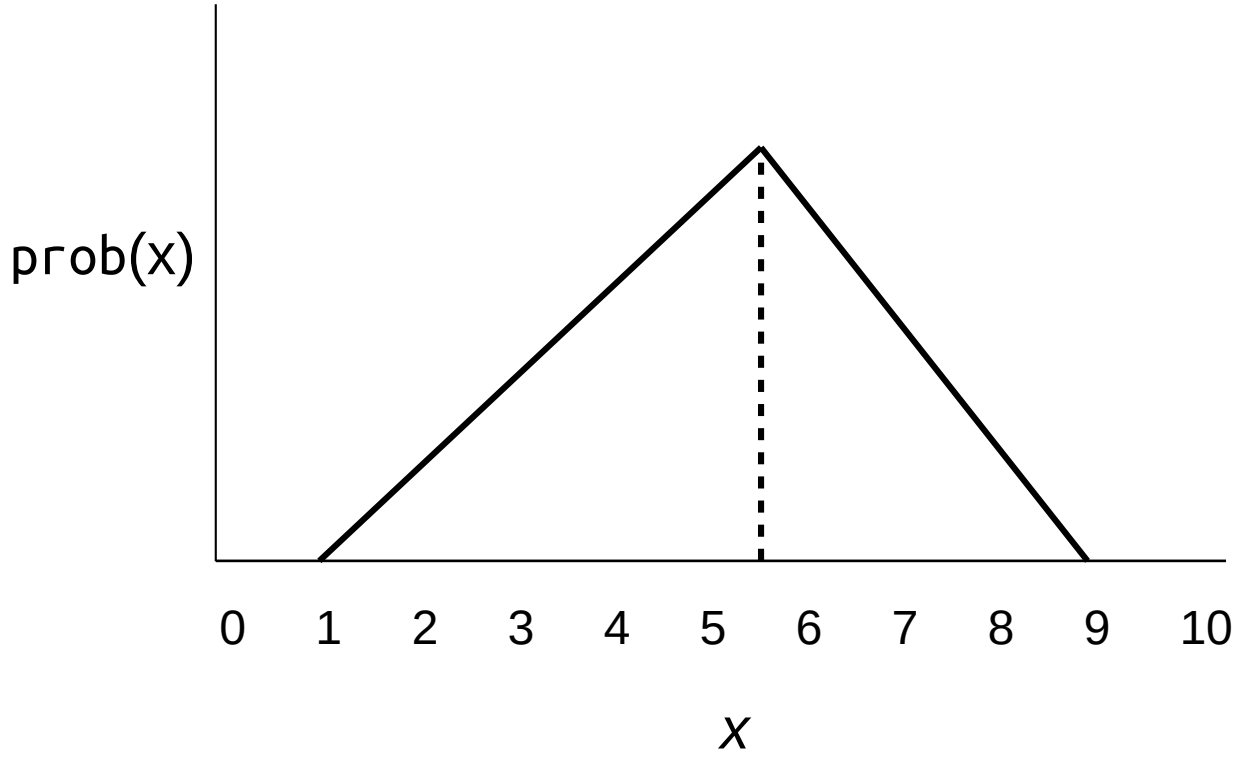


For each value in the range,
all numbers have the same probability!

About random numbers...

BUT, there are many types of random numbers
Example, **triangular** random numbers

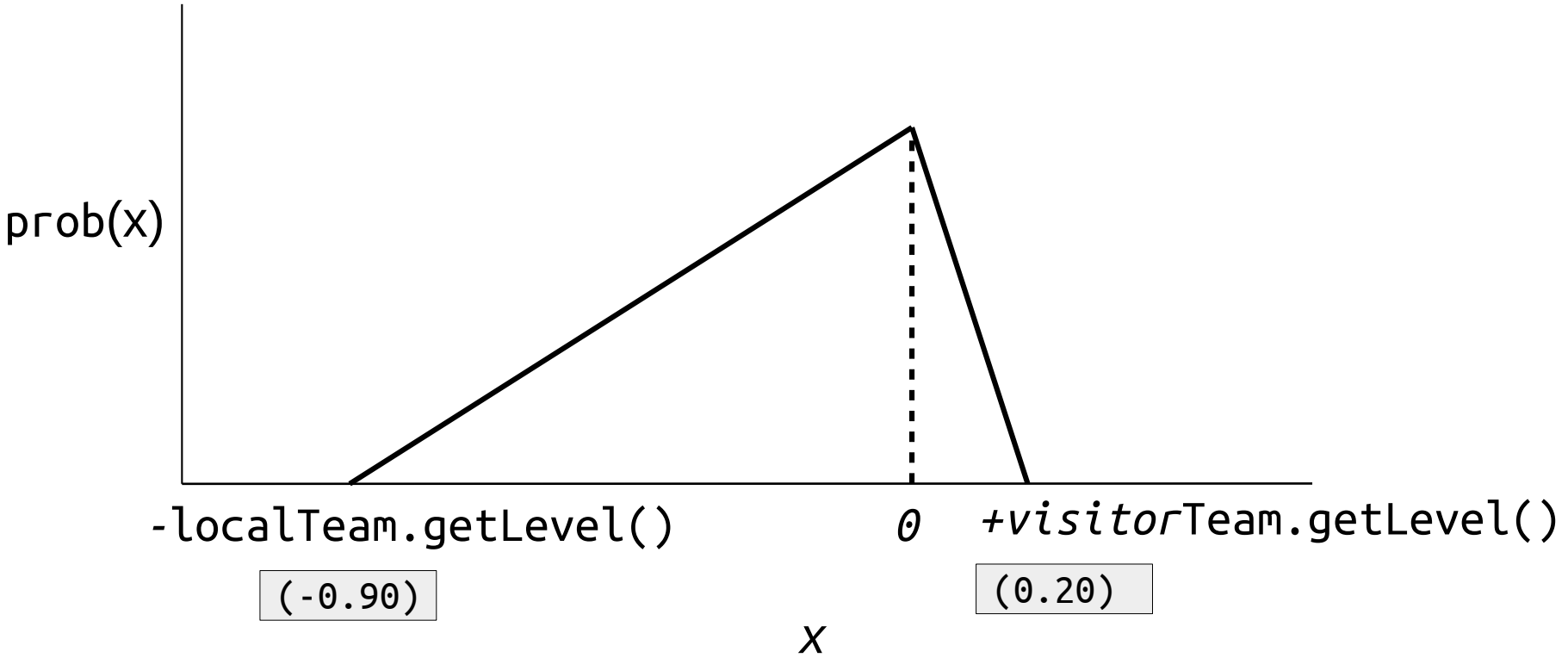
```
prob(x): probability of generating x
x: random number
```



Values close to the peak (where $x = 5.5$) will have more chance!

Triangular random numbers and our example

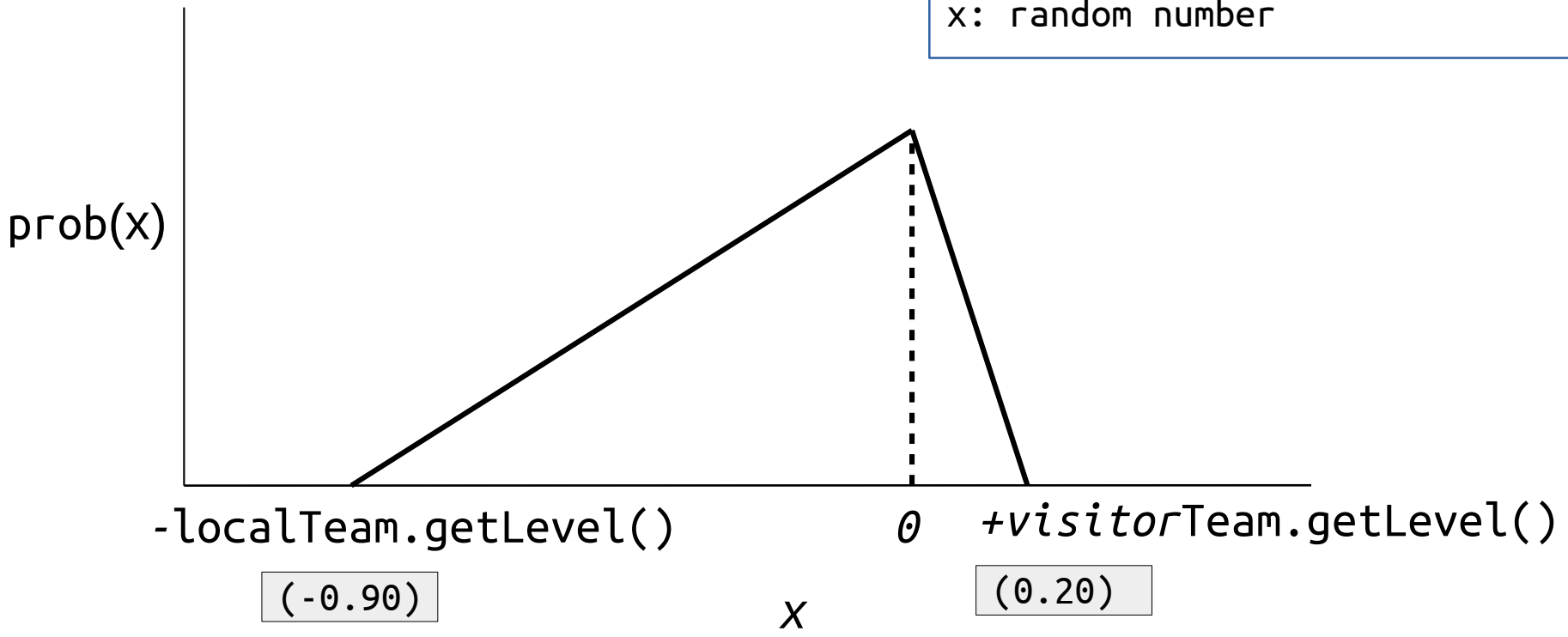
prob(x): probability of generating x
x: random number



More random numbers generated in the bigger side of the triangle!

Triangular random numbers and our example

prob(x): probability of generating x
x: random number




```
// who scored the goals?
for(int i = 0; i < numberOfGoals; i++)
{
    double x = triangular random number (-localTeam.getLevel(), 0, _visitorTeam.getLevel());


    if(x < 0) the local team scored a goal!
    else the visitor scored a goal!
}
```

Review: How to generate *uniform* random numbers in a range [a,b]?

Variant #1: C++-style

#include<random>

```cpp
#include <random>

int main()
{
    int a = 0;
    int b = 10;

    // seed
    std::random_device  r;
    std::mt19937        gen(r());

    // uniform random number generator
    std::uniform_int_distribution<int> u(a, b); // [a,b]

    int randomNumber = u(gen);

    int nextRandomNumber = u(gen);

    return 0;
}
```

Review: How to generate *uniform* random numbers in a range [a,b]?

Variant #1: C++-style
#include<random>

```cpp
#include <random>

int main()
{
    int a = 0;
    int b = 10;

    // seed
    std::random_device  r;
    std::mt19937        gen(r());

    // uniform random number generator
    std::uniform_int_distribution<int> u(a, b); // [a,b]

    int randomNumber = u(gen);

    int nextRandomNumber = u(gen);

    return 0;
}
```

Variant #2: C-style *"old-school"*
#include<ctime>, #include<cstdlib>

```cpp
#include <ctime>
#include <cstdlib>

int main()
{
    int a = 0;
    int b = 10;

    // seed
    srand(time(NULL));

    // No generator object.
    // For [a,b], use: (rand() % (b+1-a)) + a

    int randomNumber = (rand() % (b + 1 - a)) + a;

    int nextRandomNumber = (rand() % (b + 1 - a)) + a;

    return 0;
}
```

Exercise

<u>Overload</u> for the output standard operator << for structures:

(1) Match

```
m.play();

std :: cout << m << std :: endl;


localTeam=Zenit visitorTeam=Sochi stadium=Krestovsky Stadium, result=3-0
```

(2) FootballTeam

```
std :: cout << zenit << std :: endl;

name=Zenit city=Saint Petersburg wins=15 draws=0 losses=1
```

Exercise

Room 1: makes 1, 2 and 5.
Room 2: makes 3, 4 and 6
Room 3: makes 8 and 9
Room 4: makes 7 and 10

Overload for the structure `vector3D` the following operators:

1. **operator -** calculate difference between each coordinate x-y-z of two vectors, returns a vector.

2. **operator *** dot product of two vectors v1, v2 (v1.x*v2.x + v1.y*v2.y + v1.z*v2.z), returns a double.

3. **operator *** multiplication of coordinates x-y-z of a vector by a double (i.e, v*2.0), returns a vector.

4. **operator /** division of coordinates x-y-z of a vector by a double (i.e., v / 2.0), returns a vector.

5. **operator** == check if all coordinates-x-y-z of a vector are equal, returns a boolean.

6. **operator** != checks the negation of ==, returns a boolean.

7. **operator** < checks if the *magnitude* of vector v1 is strictly less than the *magnitude* of vector v2, returns a boolean.

8. **operator** > checks if the *magnitude* of vector v1 is strictly greater than the *magnitude* of vector v2, returns a boolean.

9. **operator [ ]** returns the i-th coordinate of the vector, where “i” can be 0 for x, 1, for y, 2 for z. Example: if v = (1.0, 2.0, 3.0) then v[0] = 1.0, should return a reference to a double.

10. **operator ++** post-increment v++ by 1 for each coordinate x-y-z of a vector, returns the vector itself.

Review: How to *sort* objects *by attribute*?

Variant #1
Boolean compare function

```
struct B
{
    int x;
};

bool cmp(B b1, B b2)
{
    return b1.x < b2.x;
}

using Set = std::set<B, decltype(cmp)*>;
using Vector = std::vector<B>;

int main()
{
    B b1 = {20};
    B b2 = {10};

    Set s({b1, b2}, cmp);

    Vector v = {b1, b2};
    std::sort(v.begin(), v.end(), cmp);

    return 0;
}
```

Review: How to *sort* objects *by attribute*?

Variant #1
Boolean compare function

```
struct B
{
    int x;
};

bool cmp(B b1, B b2)
{
    return b1.x < b2.x;
}

using Set = std::set<B, decltype(cmp)*>;
using Vector = std::vector<B>;

int main()
{
    B b1 = {20};
    B b2 = {10};

    Set s({b1, b2}, cmp);

    Vector v = {b1, b2};
    std::sort(v.begin(), v.end(), cmp);

    return 0;
}
```

Variant #2
Overload of operator <

```
struct B
{
    int x;
    bool operator<(B b2) const
    {
        return x < b2.x;
    }
};

using Set = std::set<B>;
using Vector = std::vector<B>;

int main()
{
    B b1 = {20};
    B b2 = {10};

    Set s = {b1, b2};

    Vector v = {b1, b2};
    std::sort(v.begin(), v.end());

    return 0;
}
```