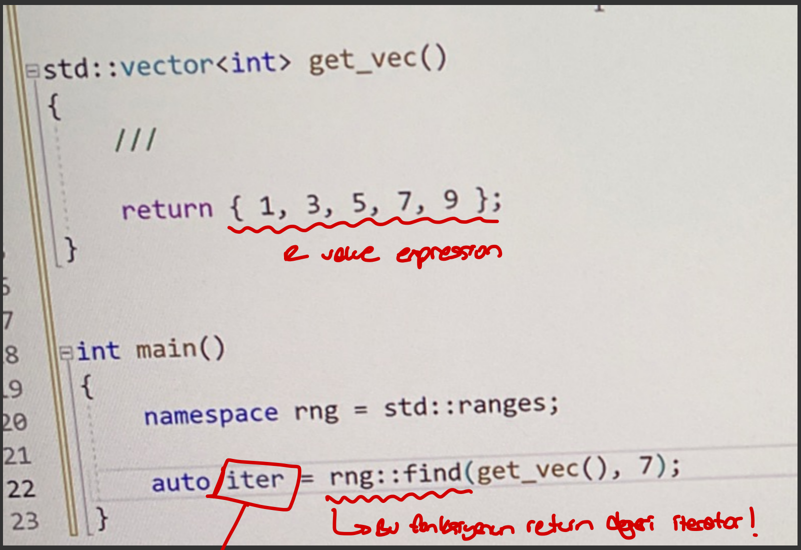
* Dangling Range:



```
std::vector<int> get_vec()
{
    ///

    return { 1, 3, 5, 7, 9 };
}

int main()
{
    namespace rng = std::ranges;

    auto iter = rng::find(get_vec(), 7);
}
```

r value expression

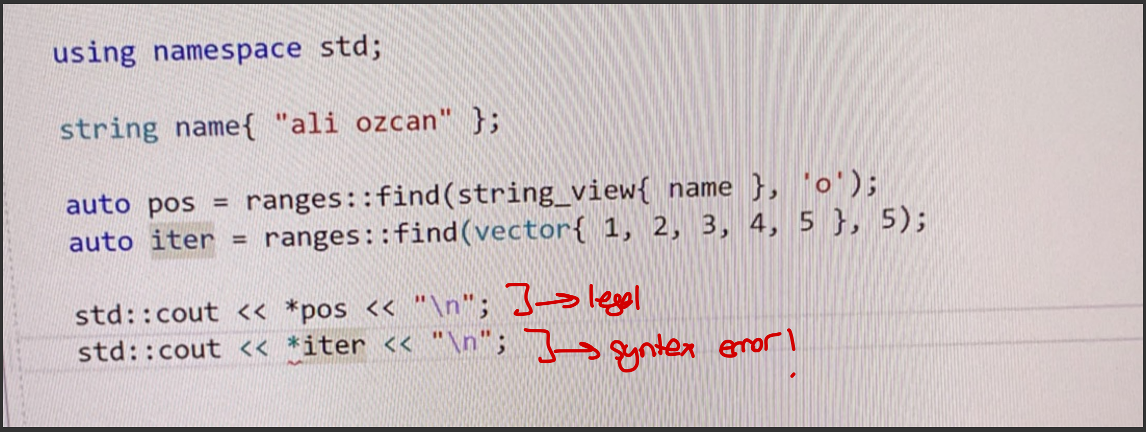
Bu fonksiyonun return değeri iterator!

range::dangling

→ Fakat range'in find fonksiyonuna R value göndermiş oluyoruz!

→ Bu tip template kodlar, universal ref kullandığı için, compile time da gönderilen ifadenin değer kategorisini tespit edip, ona göre kod return ettirebiliriz.

→ R value için, range'in dangling türünden bir struct return eder!



```
using namespace std;

string name{ "ali ozcan" };

auto pos = ranges::find(string_view{ name }, 'o');
auto iter = ranges::find(vector{ 1, 2, 3, 4, 5 }, 5);

std::cout << *pos << "\n";
std::cout << *iter << "\n";
```

→ legal

→ syntax error!

→ Bunu yapabilmek için borrowed range adında bir concept var!

→ Borrowed Range:

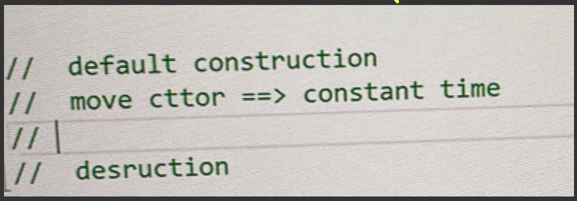
- Eğer bir range'in iteratörü, o iteratörün ilişkin olduğu range nesnesinin hayatı bitmesine karşın, hala kullanılabilirse → o bir borrowed range dir.

→ L value rangeler borrowed range dir.

→ Range'in iteratörleri range'in kendisini outlive edebilmeli! (Span / Stringview)

Contigius konteynerlar için view sınıfı!

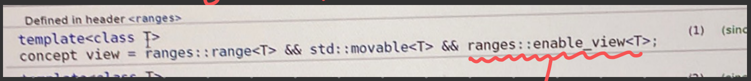
* Lightweight'ten kastımız nedir?



```
// default construction
// move cctor ==> constant time
//
// desruction
```

→ Peki compiler, özellikle bizim custom typelarımızın lightweight olduğunu nasıl anlar?

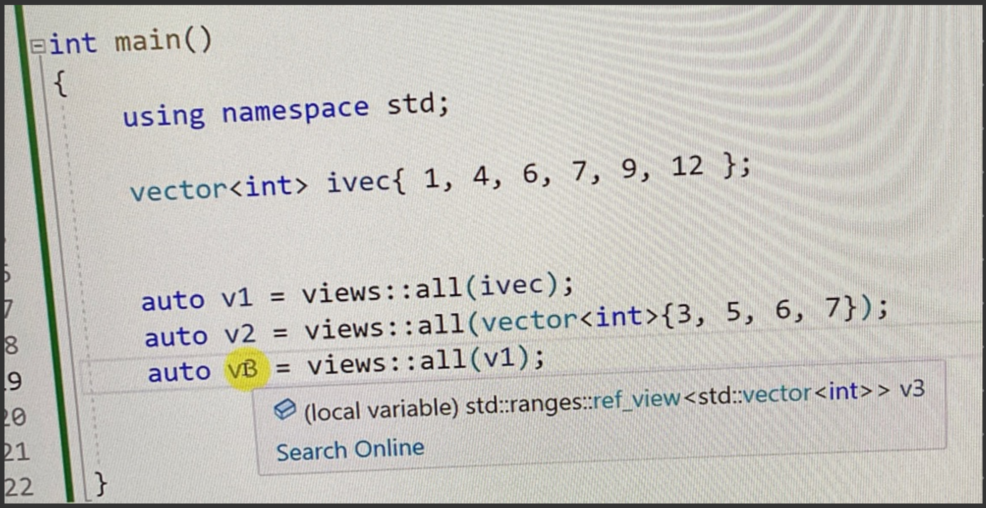
→ Direkt Anlayamaz XX



```
Defined in header <ranges>
template<class T>                                                              (1)   (sinc
concept view = ranges::range<T> && std::movable<T> && ranges::enable_view<T>;
```

view için enable view true olması gerekir!

# Tristan Brindle → borrowed range anlatan makalesi var!

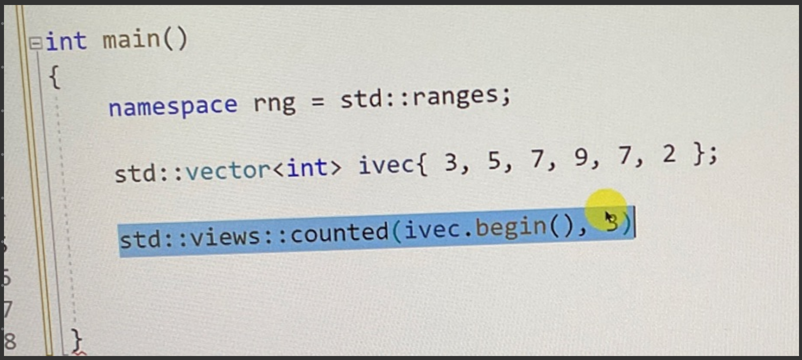


```cpp
int main()
{
    using namespace std;

    vector<int> ivec{ 1, 4, 6, 7, 9, 12 };


    auto v1 = views::all(ivec);
    auto v2 = views::all(vector<int>{3, 5, 6, 7});
    auto v3 = views::all(v1);
}
```

(local variable) std::ranges::ref_view<std::vector<int>> v3
Search Online

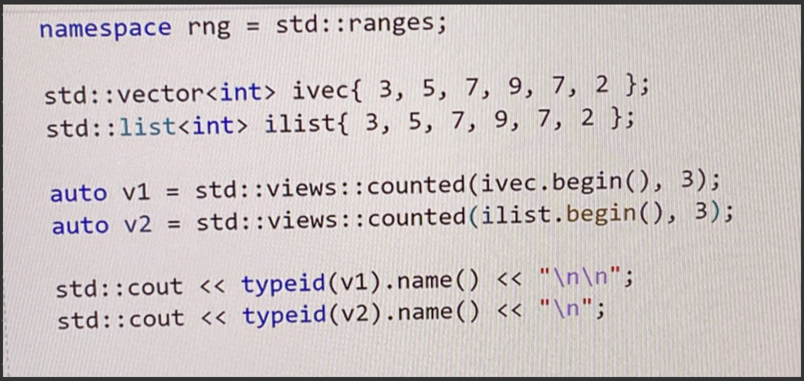
→ All ile, range'den view oluşturulabilir.

\#ranges::counted:



```cpp
int main()
{
    namespace rng = std::ranges;

    std::vector<int> ivec{ 3, 5, 7, 9, 7, 2 };

    std::views::counted(ivec.begin(), 3)

}
```

→ range'i counted ile view'a dönüştürebiliriz!

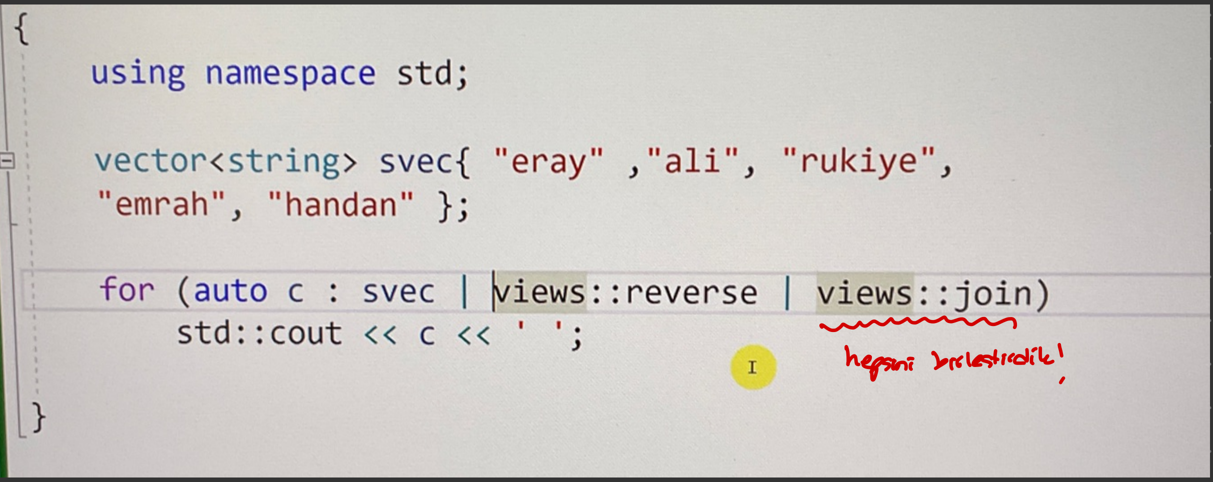


```cpp
    namespace rng = std::ranges;

    std::vector<int> ivec{ 3, 5, 7, 9, 7, 2 };
    std::list<int> ilist{ 3, 5, 7, 9, 7, 2 };

    auto v1 = std::views::counted(ivec.begin(), 3);
    auto v2 = std::views::counted(ilist.begin(), 3);

    std::cout << typeid(v1).name() << "\n\n";
    std::cout << typeid(v2).name() << "\n";
```

# Hackingcpp.com

↓

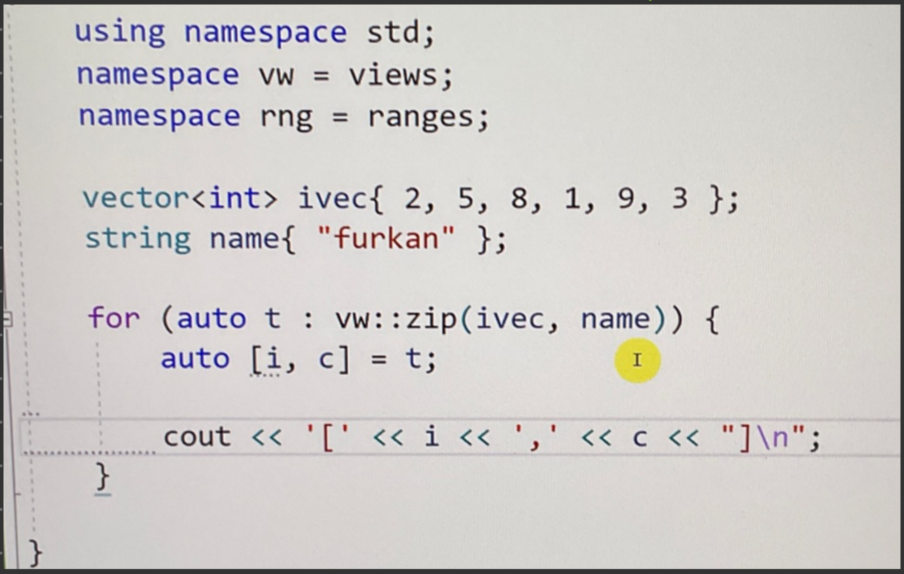
Bu siteden range viewlarına baktık!



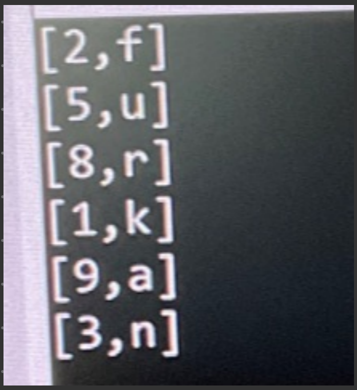
```cpp
{
    using namespace std;

    vector<string> svec{ "eray" ,"ali", "rukiye",
    "emrah", "handan" };

    for (auto c : svec | views::reverse | views::join)
        std::cout << c << ' ';

}
```

hepsini birleştirdik!

*Zip: Birden fazla range alıp, onları tuple like objectlere dönüştürür!



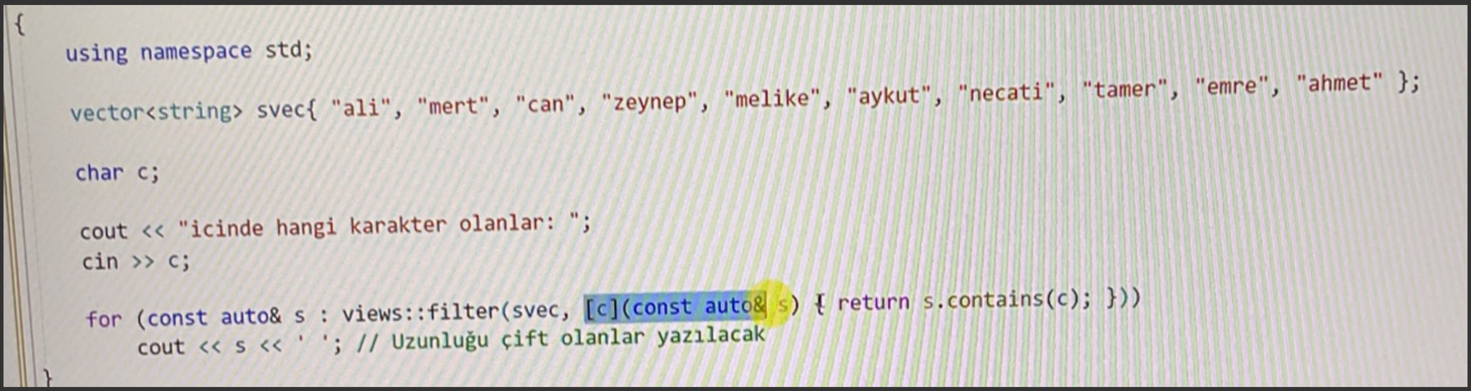
```cpp
    using namespace std;
    namespace vw = views;
    namespace rng = ranges;

    vector<int> ivec{ 2, 5, 8, 1, 9, 3 };
    string name{ "furkan" };

    for (auto t : vw::zip(ivec, name)) {
        auto [i, c] = t;

        cout << '[' << i << ',' << c << "]\n";
    }
}
```



```
[2,f]
[5,u]
[8,r]
[1,k]
[9,a]
[3,n]
```

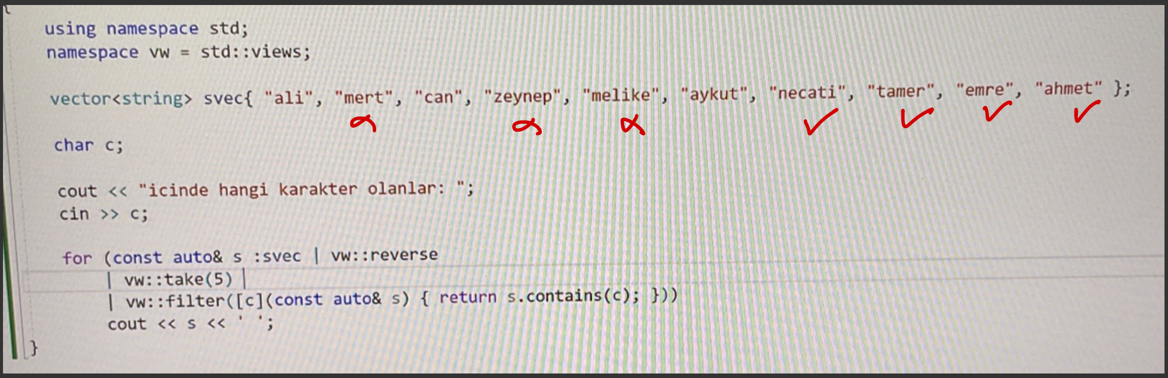
*Adaptörler ve Fabrikalar:

• Filter: unary predicate'i sağlayan öğelerden oluşan range



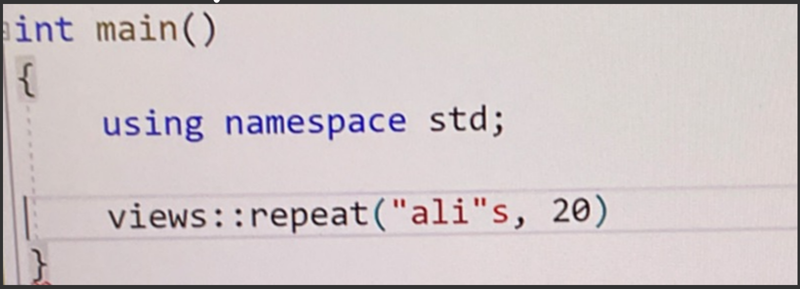
```cpp
{
    using namespace std;

    vector<string> svec{ "ali", "mert", "can", "zeynep", "melike", "aykut", "necati", "tamer", "emre", "ahmet" };

    char c;

    cout << "icinde hangi karakter olanlar: ";
    cin >> c;

    for (const auto& s : views::filter(svec, [c](const auto& s) { return s.contains(c); }))
        cout << s << ' '; // Uzunluğu çift olanlar yazılacak
}
```

• Take:



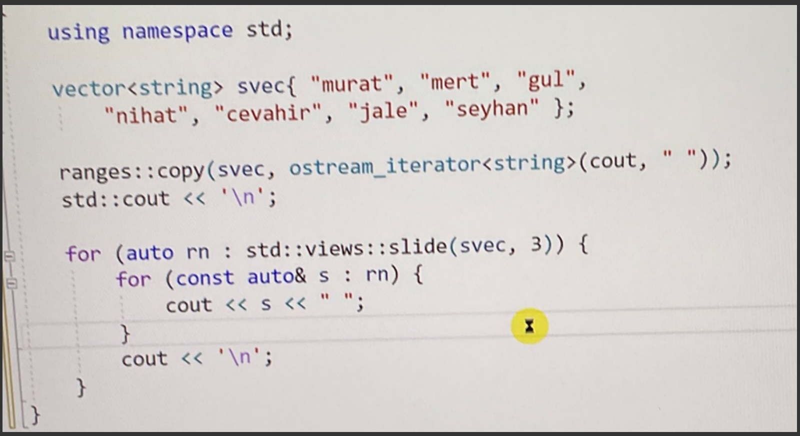
```cpp
    using namespace std;
    namespace vw = std::views;

    vector<string> svec{ "ali", "mert", "can", "zeynep", "melike", "aykut", "necati", "tamer", "emre", "ahmet" };

    char c;

    cout << "icinde hangi karakter olanlar: ";
    cin >> c;

    for (const auto& s :svec | vw::reverse
        | vw::take(5)
        | vw::filter([c](const auto& s) { return s.contains(c); }))
        cout << s << ' ';
}
```

• **Repeat:** → Bu bir fabrika fonksiyon
→ Cpp 23

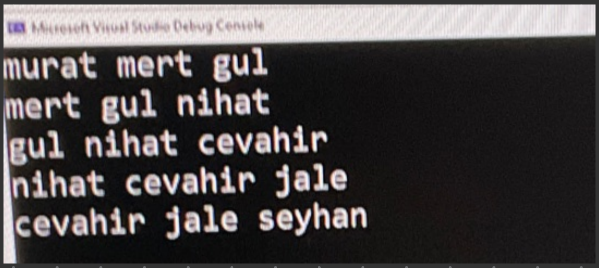


```cpp
int main()
{
    using namespace std;

    views::repeat("ali"s, 20)
}
```

• **Slide:**

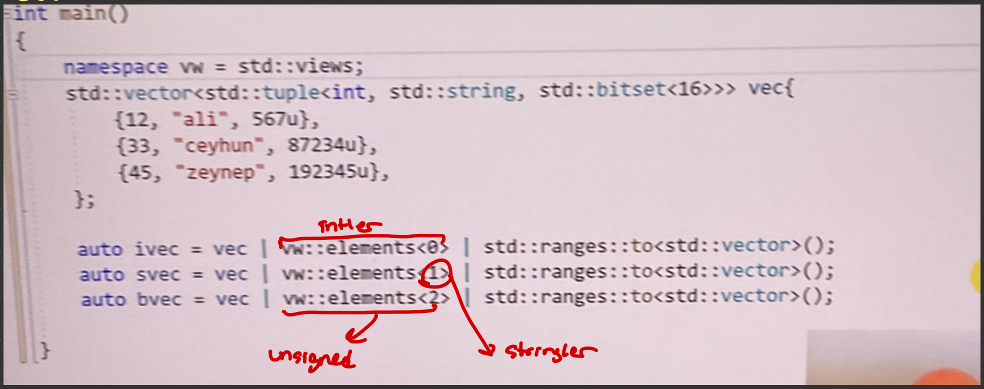


```cpp
using namespace std;

vector<string> svec{ "murat", "mert", "gul",
    "nihat", "cevahir", "jale", "seyhan" };

ranges::copy(svec, ostream_iterator<string>(cout, " "));
std::cout << '\n';

for (auto rn : std::views::slide(svec, 3)) {
    for (const auto& s : rn) {
        cout << s << " ";
    }
    cout << '\n';
}
}
```



```
murat mert gul
mert gul nihat
gul nihat cevahir
nihat cevahir jale
cevahir jale seyhan
```

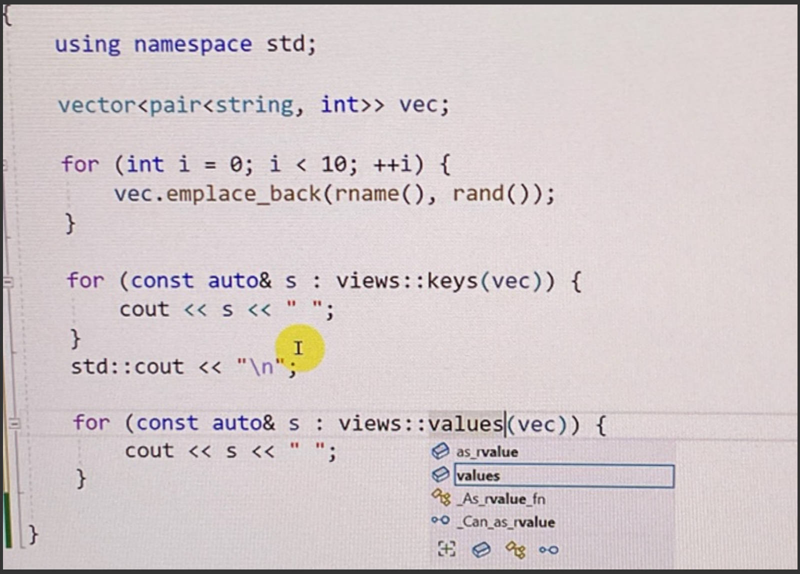
• **Elements:**



```cpp
int main()
{
    namespace vw = std::views;
    std::vector<std::tuple<int, std::string, std::bitset<16>>> vec{
        {12, "ali", 567u},
        {33, "ceyhun", 87234u},
        {45, "zeynep", 192345u},
    };

    auto ivec = vec | vw::elements<0> | std::ranges::to<std::vector>();
    auto svec = vec | vw::elements<1> | std::ranges::to<std::vector>();
    auto bvec = vec | vw::elements<2> | std::ranges::to<std::vector>();
}
```

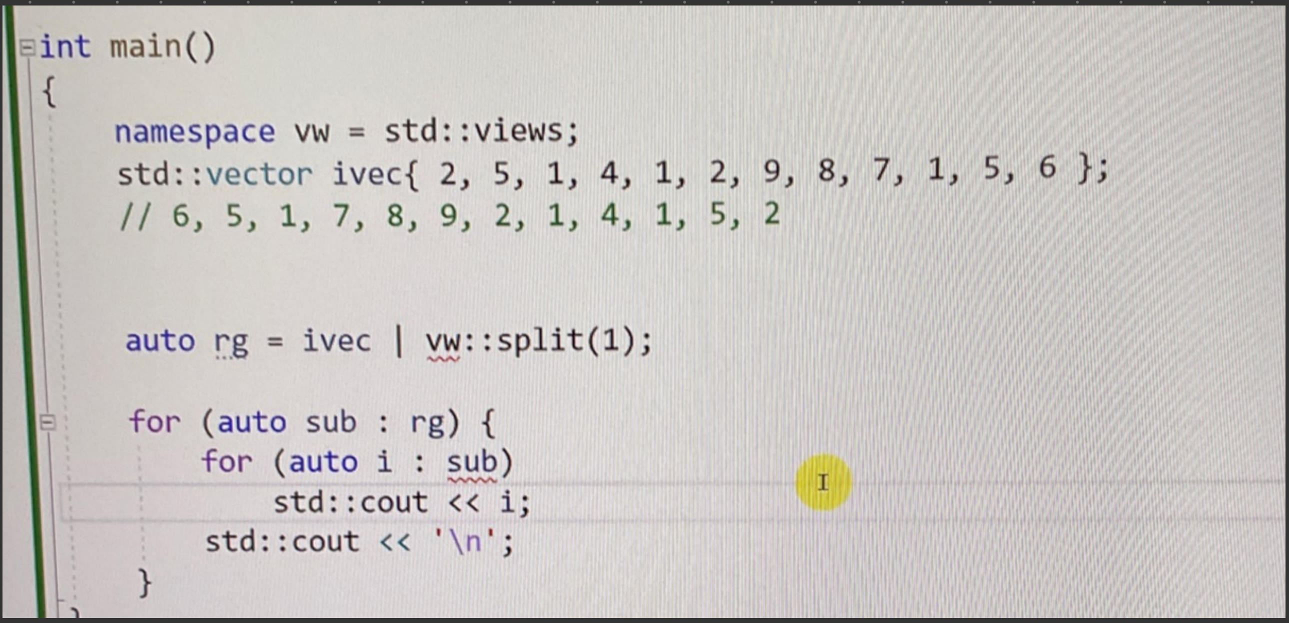
→ Tuple like objectlerin elemanlarından range oluştur!

• **Keys / Values:**



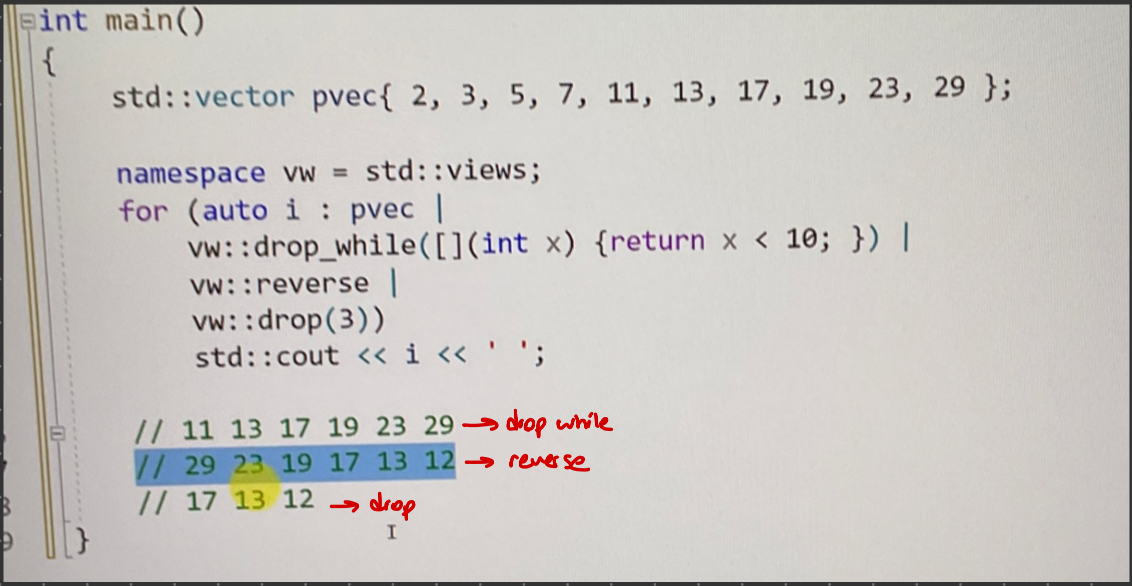
```cpp
{
    using namespace std;

    vector<pair<string, int>> vec;

    for (int i = 0; i < 10; ++i) {
        vec.emplace_back(rname(), rand());
    }

    for (const auto& s : views::keys(vec)) {
        cout << s << " ";
    }
    std::cout << "\n";

    for (const auto& s : views::values(vec)) {
        cout << s << " ";
    }
}
```

• Split: → Range'den o öğeyi ayırıyor!



```cpp
int main()
{
    namespace vw = std::views;
    std::vector ivec{ 2, 5, 1, 4, 1, 2, 9, 8, 7, 1, 5, 6 };
    // 6, 5, 1, 7, 8, 9, 2, 1, 4, 1, 5, 2


    auto rg = ivec | vw::split(1);

    for (auto sub : rg) {
        for (auto i : sub)
            std::cout << i;
        std::cout << '\n';
    }
```

• Drop_while / Drop:



```cpp
int main()
{
    std::vector pvec{ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29 };

    namespace vw = std::views;
    for (auto i : pvec |
        vw::drop_while([](int x) {return x < 10; }) |
        vw::reverse |
        vw::drop(3))
        std::cout << i << ' ';

    // 11 13 17 19 23 29 → drop while
    // 29 23 19 17 13 12 → reverse
    // 17 13 12 → drop
}
```