Ok, Ranges

Standard Ranges

What is coming in C++20?

Pythagorean Triples, Revisited

```
int main()
{
  auto const n{ 20 };
  auto const pythagorean_triples{
    [n](){
      auto const range{ std::ranges::iota_view{1, n} };
      return std::ranges::cartesian_product_view{range, range, range};
    }()
    | std::ranges::views::filter([](auto const [a, b, c]){ return a < b; })
    | std::ranges::views::filter([](auto const [a, b, c]){ return a * a + b * b == c * c; })
    | std::ranges::views::transform([](auto const [a, b, c]){ return std::tuple{a, b, c}; })
    | std::ranges::views::take_while([](auto const [a, b, c]){ return a + b + c == 1000; })
    | std::ranges::views::take(1)
    | std::ranges::views::transform([](auto const [a, b, c]){ return a * b * c; })
    | std::ranges::views::take(1)
    | std::ranges::views::transform([](auto const x){ return std::tuple{x, x * 2, x * 3}; })
    | std::ranges::views::transform([](auto const [x, y, z]){ return std::tuple{x, y, z, x + y + z}; })
    | std::ranges::views::transform([](auto const [x, y, z, s]){ return std::tuple{x, y, z, s, x * y * z}; })
    | std::ranges::views::transform([](auto const [x, y, z, s, p]){ return std::tuple{x, y, z, s, p, x / 2.0, y / 2.0, z / 2.0, s / 2.0, p / 2.0}; })
    | std::ranges::views::transform([](auto const [x, y, z, s, p, x2, y2, z2, s2, p2]){ return std::tuple{x, y, z, s, p, x2, y2, z2, s2, p2, x + y + z + s + p + x2 + y2 + z2 + s2 + p2}; })
    | std::ranges::views::transform([](auto const [x, y, z, s, p, x2, y2, z2, s2, p2, t]){ return std::tuple{x, y, z, s, p, x2, y2, z2, s2, p2, t, x * y * z * s * p * x2 * y2 * z2 * s2 * p2 * t}; })
    | std::ranges::views::transform([](auto const [x, y, z, s, p, x2, y2, z2, s2, p2, t, t2]){ return std::tuple{x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, x + y + z + s + p + x2 + y2 + z2 + s2 + p2 + t + t2}; })
    | std::ranges::views::transform([](auto const [x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3]){ return std::tuple{x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, x * y * z * s * p * x2 * y2 * z2 * s2 * p2 * t * t2 * t3}; })
    | std::ranges::views::transform([](auto const [x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4]){ return std::tuple{x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, x + y + z + s + p + x2 + y2 + z2 + s2 + p2 + t + t2 + t3 + t4}; })
    | std::ranges::views::transform([](auto const [x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5]){ return std::tuple{x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, x * y * z * s * p * x2 * y2 * z2 * s2 * p2 * t * t2 * t3 * t4 * t5}; })
    | std::ranges::views::transform([](auto const [x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6]){ return std::tuple{x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, x + y + z + s + p + x2 + y2 + z2 + s2 + p2 + t + t2 + t3 + t4 + t5 + t6}; })
    | std::ranges::views::transform([](auto const [x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7]){ return std::tuple{x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7, x * y * z * s * p * x2 * y2 * z2 * s2 * p2 * t * t2 * t3 * t4 * t5 * t6 * t7}; })
    | std::ranges::views::transform([](auto const [x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7, t8]){ return std::tuple{x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7, t8, x + y + z + s + p + x2 + y2 + z2 + s2 + p2 + t + t2 + t3 + t4 + t5 + t6 + t7 + t8}; })
    | std::ranges::views::transform([](auto const [x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7, t8, t9]){ return std::tuple{x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7, t8, t9, x * y * z * s * p * x2 * y2 * z2 * s2 * p2 * t * t2 * t3 * t4 * t5 * t6 * t7 * t8 * t9}; })
    | std::ranges::views::transform([](auto const [x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7, t8, t9, t10]){ return std::tuple{x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7, t8, t9, t10, x + y + z + s + p + x2 + y2 + z2 + s2 + p2 + t + t2 + t3 + t4 + t5 + t6 + t7 + t8 + t9 + t10}; })
    | std::ranges::views::transform([](auto const [x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11]){ return std::tuple{x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, x * y * z * s * p * x2 * y2 * z2 * s2 * p2 * t * t2 * t3 * t4 * t5 * t6 * t7 * t8 * t9 * t10 * t11}; })
    | std::ranges::views::transform([](auto const [x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12]){ return std::tuple{x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, x + y + z + s + p + x2 + y2 + z2 + s2 + p2 + t + t2 + t3 + t4 + t5 + t6 + t7 + t8 + t9 + t10 + t11 + t12}; })
    | std::ranges::views::transform([](auto const [x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13]){ return std::tuple{x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, x * y * z * s * p * x2 * y2 * z2 * s2 * p2 * t * t2 * t3 * t4 * t5 * t6 * t7 * t8 * t9 * t10 * t11 * t12 * t13}; })
    | std::ranges::views::transform([](auto const [x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14]){ return std::tuple{x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, x + y + z + s + p + x2 + y2 + z2 + s2 + p2 + t + t2 + t3 + t4 + t5 + t6 + t7 + t8 + t9 + t10 + t11 + t12 + t13 + t14}; })
    | std::ranges::views::transform([](auto const [x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15]){ return std::tuple{x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15, x * y * z * s * p * x2 * y2 * z2 * s2 * p2 * t * t2 * t3 * t4 * t5 * t6 * t7 * t8 * t9 * t10 * t11 * t12 * t13 * t14 * t15}; })
    | std::ranges::views::transform([](auto const [x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15, t16]){ return std::tuple{x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15, t16, x + y + z + s + p + x2 + y2 + z2 + s2 + p2 + t + t2 + t3 + t4 + t5 + t6 + t7 + t8 + t9 + t10 + t11 + t12 + t13 + t14 + t15 + t16}; })
    | std::ranges::views::transform([](auto const [x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15, t16, t17]){ return std::tuple{x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15, t16, t17, x * y * z * s * p * x2 * y2 * z2 * s2 * p2 * t * t2 * t3 * t4 * t5 * t6 * t7 * t8 * t9 * t10 * t11 * t12 * t13 * t14 * t15 * t16 * t17}; })
    | std::ranges::views::transform([](auto const [x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15, t16, t17, t18]){ return std::tuple{x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15, t16, t17, t18, x + y + z + s + p + x2 + y2 + z2 + s2 + p2 + t + t2 + t3 + t4 + t5 + t6 + t7 + t8 + t9 + t10 + t11 + t12 + t13 + t14 + t15 + t16 + t17 + t18}; })
    | std::ranges::views::transform([](auto const [x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15, t16, t17, t18, t19]){ return std::tuple{x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15, t16, t17, t18, t19, x * y * z * s * p * x2 * y2 * z2 * s2 * p2 * t * t2 * t3 * t4 * t5 * t6 * t7 * t8 * t9 * t10 * t11 * t12 * t13 * t14 * t15 * t16 * t17 * t18 * t19}; })
    | std::ranges::views::transform([](auto const [x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15, t16, t17, t18, t19, t20]){ return std::tuple{x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15, t16, t17, t18, t19, t20, x + y + z + s + p + x2 + y2 + z2 + s2 + p2 + t + t2 + t3 + t4 + t5 + t6 + t7 + t8 + t9 + t10 + t11 + t12 + t13 + t14 + t15 + t16 + t17 + t18 + t19 + t20}; })
    | std::ranges::views::transform([](auto const [x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15, t16, t17, t18, t19, t20, t21]){ return std::tuple{x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15, t16, t17, t18, t19, t20, t21, x * y * z * s * p * x2 * y2 * z2 * s2 * p2 * t * t2 * t3 * t4 * t5 * t6 * t7 * t8 * t9 * t10 * t11 * t12 * t13 * t14 * t15 * t16 * t17 * t18 * t19 * t20 * t21}; })
    | std::ranges::views::transform([](auto const [x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15, t16, t17, t18, t19, t20, t21, t22]){ return std::tuple{x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15, t16, t17, t18, t19, t20, t21, t22, x + y + z + s + p + x2 + y2 + z2 + s2 + p2 + t + t2 + t3 + t4 + t5 + t6 + t7 + t8 + t9 + t10 + t11 + t12 + t13 + t14 + t15 + t16 + t17 + t18 + t19 + t20 + t21 + t22}; })
    | std::ranges::views::transform([](auto const [x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15, t16, t17, t18, t19, t20, t21, t22, t23]){ return std::tuple{x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15, t16, t17, t18, t19, t20, t21, t22, t23, x * y * z * s * p * x2 * y2 * z2 * s2 * p2 * t * t2 * t3 * t4 * t5 * t6 * t7 * t8 * t9 * t10 * t11 * t12 * t13 * t14 * t15 * t16 * t17 * t18 * t19 * t20 * t21 * t22 * t23}; })
    | std::ranges::views::transform([](auto const [x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15, t16, t17, t18, t19, t20, t21, t22, t23, t24]){ return std::tuple{x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15, t16, t17, t18, t19, t20, t21, t22, t23, t24, x + y + z + s + p + x2 + y2 + z2 + s2 + p2 + t + t2 + t3 + t4 + t5 + t6 + t7 + t8 + t9 + t10 + t11 + t12 + t13 + t14 + t15 + t16 + t17 + t18 + t19 + t20 + t21 + t22 + t23 + t24}; })
    | std::ranges::views::transform([](auto const [x, y, z, s, p, x2, y2, z2, s2, p2, t, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15, t16, t17, t18, t19, t20, t21, t22, t23, t24, t25]){ return std
```

Fundamental Concepts

Iterator and Range Concepts

New Convenience Iterator Traits

Safe Range Access Functions

Proxies and Proxy Iterator Support

Contiguous Iterator Support

Constrained Algorithms

Range Algorithms

Constrained Function Objects

Generalized Callables

Projections

Range Utilities

view_interface

subrange

all_view

Range Generators

Range Adaptors

Future Directions

Summary