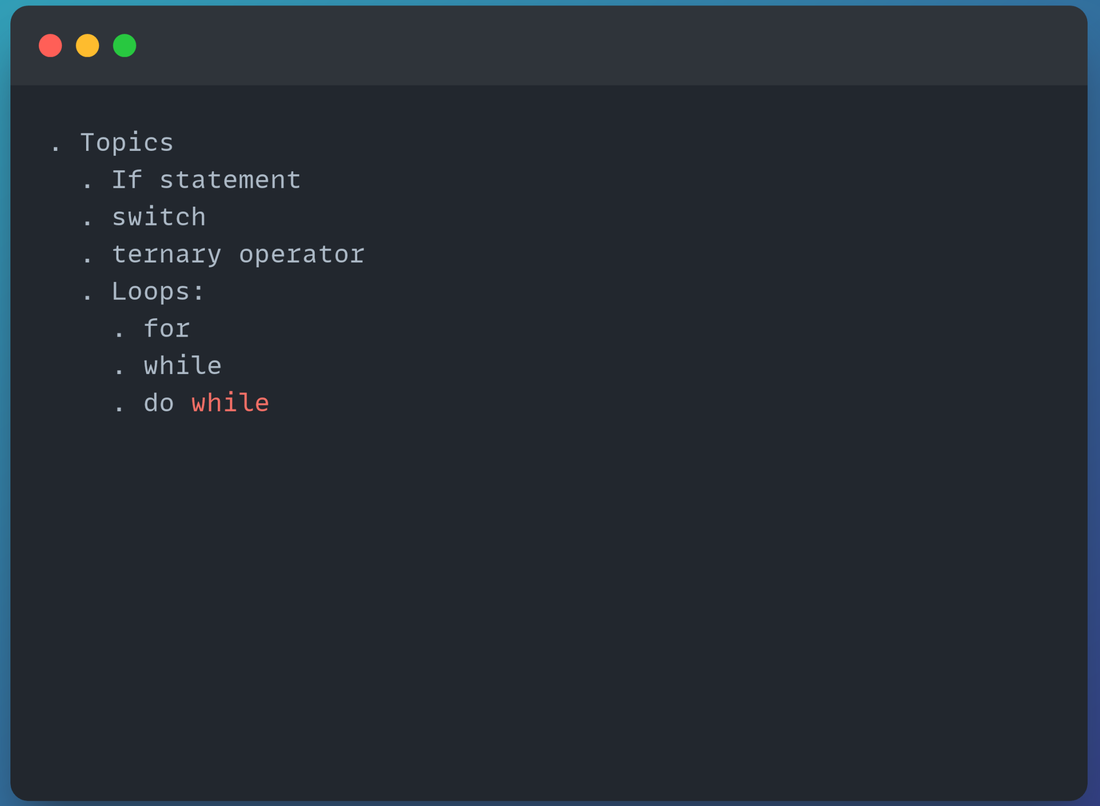
Flow control



if statement

```
// If statement
int number1{ 75 };
int number2{ 60 };
bool result = (number1 < number2);// Expression yielding the condition


//Free standing if statement
fmt::println( "free standing if statement" );
//if(result){
    if(result == true){
    fmt::println("{} is less than {}" , number1, number2 );
}

//if(!result){
    if(!(result == true)){
    fmt::println( " {}is NOT less than{} " , number1,number2 );
}
```

if...else

```
if(result == true){
    fmt::println(" {} is less than {}" , number1, number2 );
}else{
    fmt::println( "{} is NOT less than {}" , number1, number2 );
}
```

expression as a condition

```
if(number1 < number2){
    fmt::println("{} is less than {}" ,number1, number2 );
}else{
    fmt::println("{} is NOT less than {}" ,number1, number2 );
}
```

nested if statements

```
bool red = false;
bool green{ true };
bool yellow{ false };
bool police_stop{ true };

 //   If green : go
 //   If red, yellow : stop
 //   If green and police_stop : stop

if(red){
    fmt::println( "Stop" );
}
if(yellow){
    fmt::println( "Slow down" );
}
if(green){
    fmt::println( "Go" );
}

fmt::println( "Police officer stops(verbose)" );
if(green){
  if(police_stop){
      fmt::println( "Stop" );
  }else{
        fmt::println( "Go" );
     }
 }
```

nested if statements

```
bool red = false;
bool green{ true };
bool yellow{ false };
bool police_stop{ true };

 //   If green : go
 //   If red, yellow : stop
 //   If green and police_stop : stop

if(red){
    fmt::println( "Stop" );
}
if(yellow){
    fmt::println( "Slow down" );
}
if(green){
    fmt::println( "Go" );
}

fmt::println( "Police officer stops(verbose)" );
if(green){
  if(police_stop){
      fmt::println( "Stop" );
  }else{
        fmt::println( "Go" );
     }
 }
```

nested if statements

```
if (green && !police_stop) {
  fmt::println("Go");
} else {
  fmt::println("Stop");
}
```

if with initializer

```
bool go{ true };
if (int speed{ 10 }; go) {
  fmt::println("speed : {}", speed);

  if (speed > 5) {
    fmt::println("Slow down!");
  } else {
    fmt::println("All good!");
  }
} else {
  fmt::println("speed : {}", speed);
  fmt::println("Stop");
}
```

else if

```
// Tools
const int Pen{ 10 };
const int Marker{ 20 };
const int Eraser{ 30 };
const int Rectangle{ 40 };
const int Circle{ 50 };
const int Ellipse{ 60 };

int main(){

    int tool{ Eraser };

    if (tool == Pen) {
      fmt::println("Active tool is pen");
      // Do the actual painting
    } else if (tool == Marker) {
      fmt::println("Active tool is Marker");
    } else if (tool == Eraser) {
      fmt::println("Active tool is Eraser");
    } else if (tool == Rectangle) {
      fmt::println("Active tool is Rectangle");
    } else if (tool == Circle) {
      fmt::println("Active tool is Circle");
    } else if (tool == Ellipse) {
      fmt::println("Active tool is Ellipse");
    }

}
```

switch

```cpp
// Tools
const int Pen{ 10 };
const int Marker{ 20 };
const int Eraser{ 30 };
const int Rectangle{ 40 };
const int Circle{ 50 };
const int Ellipse{ 60 };

int main(){
    int tool{ Eraser };

  switch (tool) {
    case Pen: {
      fmt::println("Active tool is Pen");
    } break;

    case Marker: {
      fmt::println("Active tool is Marker");
    } break;

    case Eraser:
    case Rectangle:
    case Circle: {
      fmt::println("Drawing Shapes");
    } break;

    default: {
      fmt::println("No match found");
    } break;
  }
}
```

switch with initializer

```
// Tools
const int Pen{ 10 };
const int Marker{ 20 };
const int Eraser{ 30 };
const int Rectangle{ 40 };
const int Circle{ 50 };
const int Ellipse{ 60 };

int main(){
    int tool{ Eraser };
    switch (double strength{ 3.56 }; tool) {
        case Pen: {
            fmt::println("Active tool is Pen. strength : {}", strength);
        } break;

        case Eraser:
        case Rectangle:
        case Circle: {
            fmt::println("Drawing Shapes. strength : {}", strength);
        } break;

        case Ellipse: {
            fmt::println("Active tool is Ellipse. strength : {}", strength);
        } break;

        default: {
            fmt::println("No match found. strength : {}", strength);
        } break;
    }
}
```

short circuit evaluations

```
bool a{ true };
bool b{ true };
bool c{ true };
bool d{ false };

bool p{ false };
bool q{ false };
bool r{ false };
bool m{ true };


//AND: If one of the operands is false, the result is false.
fmt::println( "AND short circuit" );
bool result = a && b && c && d;
fmt::println( "AND - result : {}", result );

//OR: If one of the operands is true, the result is true.
fmt::println( "OR short circuit" );
result = p || q || r || m;
fmt::println( "OR - result : {}", result );
```

short circuit evaluations

```
bool car()
{
  //fmt::println("car function running");
  return false;
}

bool house(){ return true; }
bool job(){ return false; }
bool spouse(){ return false ;}

int main(){

   if (car() && house() && job() && spouse()) {
       fmt::println( "AND - I am happy" );
   }
   else {
       fmt::println( "AND - I am sad" );
   }

  if (car() || house() || job() || spouse()) {
    fmt::println("OR - I am happy");
  } else {
    fmt::println("OR - I am sad");
  }
}
```

ternary operator

```rust
  int max{};

int a{ 35 };
int b{ 200 };

//  if(a >  b){
//     max = a;
//  }else{
//        max = b;
//  }

max = (a > b) ? a : b;// Ternary operator

fmt::println("max :{} ", max);
```

std::unreachable (C++23)

std::unreachable() is a function introduced as part of <utility> to mark code paths that should never be reached. It's used to inform the compiler that certain parts of code are logically unreachable, which can lead to optimization opportunities.

```cpp
//std::unreachable: C++ 23
void handleColor(int color) {
    switch (color) {
        case Red:
            fmt::println("Handling Red");
            break;
        case Green:
            fmt::println("Handling Green");
            break;
        case Blue:
            fmt::println("Handling Blue");
            break;
        default:
            // We expect that all cases are covered,
            // so reaching here should never happen.
            std::unreachable();
    }
}

int main(){

    handleColor(Red);
    handleColor(Green);
    handleColor(Blue);

}
```