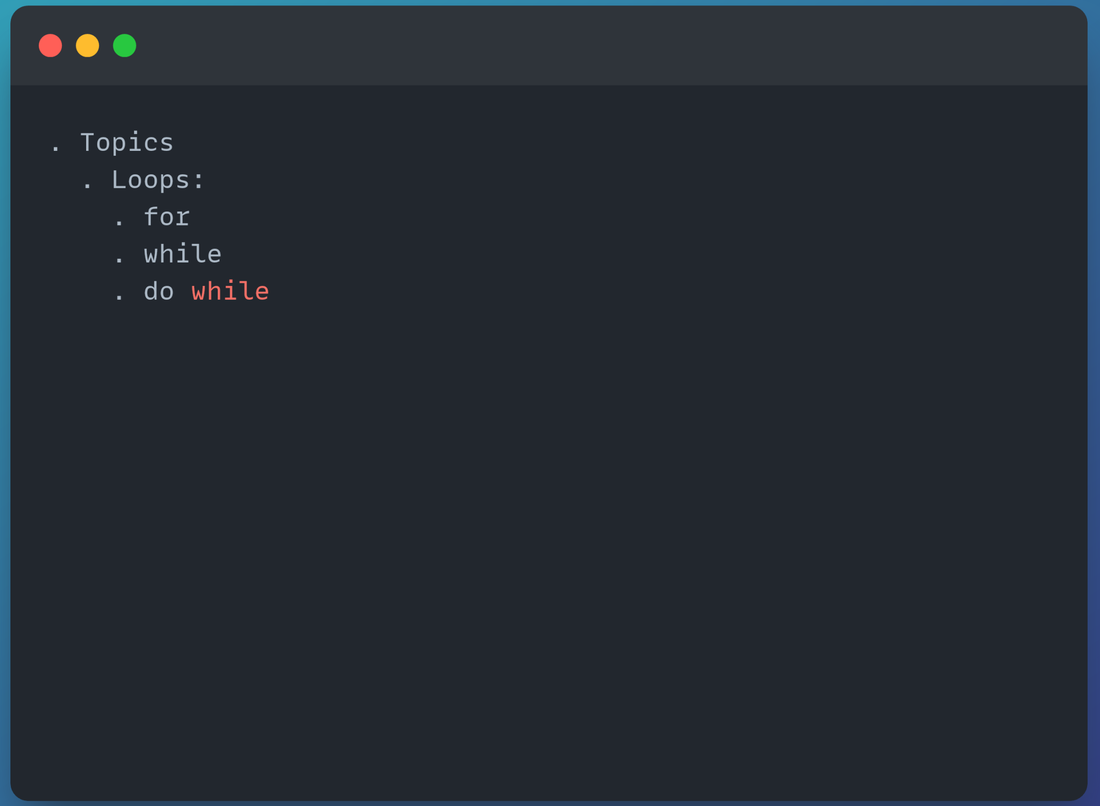
Loops



Rationale

```cpp
// Print I love C++ 10 times: The bad way
fmt::println( "I love C++" );
fmt::println( "I love C++" );
fmt::println( "I love C++" );
fmt::println( "I love C++" );
fmt::println( "I love C++" );
fmt::println( "I love C++" );
fmt::println( "I love C++" );
fmt::println( "I love C++" );
fmt::println( "I love C++" );
fmt::println( "I love C++" );
```

for loop

```
/*
    Loop components:
      . initializer
      . end condition
      . incrementation
      . body
*/
//IMPORTANT: Run this through a debugger


// for loop : the good way
for( unsigned int i{0} ; i < 100 ;++i){
  //Whatever we want the loop to run
  fmt::println("{} : I love C++", i );
}
```

for loop with size_t

```
/*
    Loop components:
      . initializer
      . end condition
      . incrementation of the iterator
      . body
*/
//IMPORTANT: Run this through a debugger


// Use size_t : a representation of some unsigned int for positive numbers [sizes]
for(size_t i{0} ; i < 10 ; ++i){
  fmt::println( " {} : I love C++", i );
}
fmt::println( "Loop done!" );
//sizeof(size_t)
fmt::println( "sizeof(size_t): {}" , sizeof(size_t) );
```

Scope of the iterator

```
/*
    Loop components:
      . initializer
      . end condition
      . incrementation of the iterator
      . body
*/
//IMPORTANT: Run this through a debugger


// Scope of the iterator
for(size_t i{0} ; i < 10 ; ++i){
  fmt::println( "{} : I love C++", i );
}
fmt::println( "Loop done!" );
fmt::println( "i : {}", i ); //Compiler error : i is not in scope
```

Leave out iterator declaration

```cpp
// Leave out the iterator declaration part
size_t i{0}; // Iterator defined outside

for(  ; i < 10 ; ++i){
  fmt::println( " {}: I love C++", i );
}
fmt::println( "Loop done!" );
fmt::println( "i : {}",  i );
```

Don't hardwire values in: Magic numbers are BAD!

```cpp
// Don't hard code values : BAD!
constexpr size_t COUNT{ 100 };

for (size_t i{ 0 }; i < COUNT; ++i) { fmt::println(" {}: I love C++", i); }
fmt::println("Loop done!");
```

Range based for loop

```cpp
//Range based for loop
std::vector <int> values{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };// This is a collection of integers
// The variable value will be assigned a value from the values array on each iteration
for(size_t i {0} ; i < 10 ; ++i){
  fmt::println( "value: {}" , values[i] );
}

for (int value : values){
  //value holds a copy of the current iteration in the whole collection
  fmt::println( " value: {}" , value );
}
```

Auto type deduction

```
// Auto type deduction
for (auto value : { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }) {
  // value holds a copy of the current iteration in the whole bag
  fmt::println(" value : {}", value);
}
```

While loop

```cpp
//3. While loop
constexpr size_t COUNT{ 100 };
size_t i{ 0 };// Iterator declaration

while (i < COUNT) {// Test
  fmt::println("{} : I love C++", i);
  ++i;// Incrementation
}
fmt::println("Loop done!");
```

Do while loop

```cpp
//Do while loop
constexpr int COUNT{ 0 };
size_t i{ 0 };// Iterator declaration

do {
  fmt::println("{} : I love C++", i);
  ++i;// Incrementation
} while (i < COUNT);

fmt::println("Loop done!");
```

Infinite loops

```cpp
// Infinite loop : for loop
for(size_t i{};true ; ++i){
  fmt::println( "{} : I love C++", i);
}

// Infinite loop : while loop
size_t i{0};

while(true){
fmt::println( "{} : I love C++", i);
  ++i;
}

// Infinite loop : do while loop
size_t i{ 0 };

do {
  fmt::println("{} : I love C++", i);
  ++i;
} while (true);
```

Decrementing loops

```
//6. Decrementing loops
constexpr size_t COUNT{ 5 };

// For loops
for(size_t i{COUNT} ; i > 0 ; --i){
    fmt::println( "i : {}", i );
  }
```

Break and continue

```cpp
// break and continue : for loops
constexpr size_t COUNT{20};

for(size_t i{0} ; i < COUNT ; ++i ){

   if(i==5)
      continue;

   if(i == 11)
      break; // Breaks out of the loop
fmt::println("i: {}", i );

}
fmt::println( "Loop done!");
```

```
i: 0
i: 1
i: 2
i: 3
i: 4
i: 6
i: 7
i: 8
i: 9
i: 10
Loop done!
```