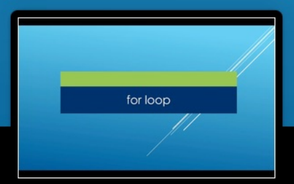
for loop

Repetitive tasks

```
//Print I love C++ 10 times
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
```



for loop

```cpp
for(unsigned int i{}; i < 10 ;++i){
    std::cout << "I love C++" << std::endl;
}
```

Pillars of any loop

- Iterator
- Starting Point
- Test(controls when the loop stops)
- Increment(Decrement)
- Loop body

size_t	Not a type, just a type alias for some unsigned int representation

Other operations in the loop body

```
for( size_t i{0} ; i < 10 ; ++i){
    std::cout << "i : " << i <<  ". Double that and you get " << 2*i << std::endl;
}
```

Can leave out the {}

```
for (size_t i{} ; i < 5 ; ++i)
    std::cout << "Single statement in body. Can leave out {} on loop boby" << std::endl;
```

Scope of the iterator

```
int main(int argc, char **argv)
{
    for (size_t i{0} ; i < 10 ; ++i){
        // i is valid to use within the boundaries of the {} here
        std::cout << "i is usable here, the value is : " << i << std::endl;
    }
    //If you try to access i here, you'll get an error.
    //i doesn't exist in the main function local scope

    return 0;
}
```

Iterator can live outside the loop scope

```
size_t j{} ;

for(j ; j < 10 ; ++j){
    std::cout << "Using the j variable from main function local scope : " << j << std::endl;
}
std::cout << "Loop done, the value of j is : " << j << std::endl;
```

Hard coded values are bad

```
const size_t COUNT {10};
for(size_t j{} ; j < COUNT ; ++j){
    std::cout << "The value of j is : " << j << std::endl;
}
```