Functions: Parameters

```
/*
    . Passing parameters to functions:
        . By value:
            . The function receives a copy of the value passed to it.
            . The original value is not modified.

        . By reference:
            . The function receives a reference to the value passed to it.
            . The original value is modified.

        . By Pointer:
            . The function receives a pointer to the value passed to it.
            . The original value is modified.

        . Default parameters:
            . Parameters that have default values.
            . They only have to show up in the declaration.
*/
```

Parameters by value

```cpp
void say_age(int age)    // Parameter
{
    ++age;
    fmt::println("Hello, you are {} years old! &age: {}", age, fmt::ptr(&age));
}

int main(){

    //Pass by value
    int age{ 23 };// Local
    fmt::println("age (before call): {} &age: {}", age, fmt::ptr(&age));
    say_age(age);// Argument
    fmt::println("age (after call): {} &age: {}", age, fmt::ptr(&age));


}
```

Parameters by const value

```
//Pass by const value
void say_age(const int age)
{
  //++age;  //Can't assign to a variable that is const
  fmt::println("Hello, you are {} years old! &age: ", age, fmt::ptr(&age));
}


int main(){

    int age{ 23 };// Local
    fmt::println("age (before call): {} &age: {}", age, fmt::ptr(&age));
    say_age(age);// Argument
    fmt::println("age (after call): {} &age: ", age, fmt::ptr(&age));


}
```

Parameters by pointer

```cpp
//Pass by pointer
void say_age(int *age)
{// Parameter
  ++(*age);
  fmt::println("Hello ,you are {} years old! &age: {}", *age, fmt::ptr(&age));// 24
}

int main(){

    //Pass by pointer
    int age{ 23 };// Local
    fmt::println("age (before call), {} &age: {} ", age, fmt::ptr(&age));// 23
    say_age(&age);// Argument
    fmt::println("age (after call), {} &age: {}", age, fmt::ptr(&age));// 24

}
```

Parameters by pointer to const

```
//Pass by pointer to const
int dog_count{ 10 };// Global
void say_age(const int *age)    //The value pointed to is constant
{
    //++(*age); //One way to do this

    //Another way to do this
    // int value = *age;
    // ++value;
    // *age = value;
    fmt::println("Hello , you are {}  years old! &age :{} ", *age, fmt::ptr(&age));// 24

    //But we can make the pointer point somewhere else
    age = &dog_count; //This compiles
}

int main(){

    //Pass by pointer to const
    int age{ 23 };// Local
    fmt::println("age (before call): {} &age: {}", age, fmt::ptr(&age));// 23
    say_age(&age);// Argument
    fmt::println("age (after call): {} &age: {}", age, fmt::ptr(&age));// 24

}
```

Parameters by const pointer to const

```cpp
//Pass by const pointer to const
int dog_count{ 10 };// Global
void say_age(const int *const age)
{
    //++(*age); // The value pointed to is const.
    fmt::println("Hello , you are {} years old! &age : {}", *age, fmt::ptr(&age));// 24
    //age = & dog_count; // The pointer itself is const, you can't make it point somewhere else.
}


int main(){

    //Pass by const pointer to const
    int age{ 23 };// Local
    fmt::println("age (before call): {} &age: {}", age, fmt::ptr(&age));
    say_age(&age);// Argument
    fmt::println("age (after call): {} &age: {}", age, fmt::ptr(&age));

}
```

Parameters by reference

```
//Pass by reference
void say_age(int &age)
{
  ++age;
  fmt::println("Hello, you are {} years old! &age: {}", age, fmt::ptr(&age));// 24
}

int main(){
  //Pass by reference
  int age{ 23 };// Local
  fmt::println("age (before call): {} &age: {}", age, fmt::ptr(&age));
  say_age(age);// Argument
  fmt::println("age (after call): {} &age: {}", age, fmt::ptr(&age));

}
```

Parameters by const reference

```cpp
//Pass by const reference
void say_age(const int &age)
{
  //++age;
  fmt::println("Hello, you are {} years old! &age: {}", age, fmt::ptr(&age));// 24
}

int main(){
  //Pass by const reference
  int age{ 23 };// Local
  fmt::println("age (before call) : {} &age : {}", age, fmt::ptr(&age));
  say_age(age);// Argument
  fmt::println("age (after call) : {} &age : {}", age, fmt::ptr(&age));

}
```

Functions: Default parameters

```
//Default parameters: They only have to show up in the declaration.
void compute(int age = 27, double weight = 72.4, double distance = 12.5){
  fmt::println("Doing computations on age : {} weight :{} and distance :{}  ", age, weight, distance);
}

void greet_teacher(
        std::string_view name_sv = "teacher",
        int homeworks = 12,
        int exams = 4,
        double pass_rate = 0.5,
        std::string_view first_dpmt = "Computer Sce"){
  fmt::println("Good morning {} !", name_sv);
  fmt::println("In the past semester, we had {} homeworks, and {} exams. The pass rate was around {} ",
    homeworks,
    exams,
    pass_rate);
  fmt::println("The best performing department is {} ", first_dpmt);
}
```