Ternary Operator ?:

In this lesson, we will see what a ternary operator does and how we can use it in our code.

WE'LL COVER THE FOLLOWING

Ternary operator `?:`

The `?:` operator works very similarly to an `if-else` statement:

```
if (/* condition check */) {
  /* ... expression(s) to execute if true */
} else {
  /* ... expression(s) to execute if false */
}
```

The `if` statement executes either the block for the case of true or the block for the case of false. As you remember, being a statement, it does not have a value; `if` merely affects the execution of code blocks.

On the other hand, the `?:` operator is an expression. In addition to working similarly to the `if-else` statement, it produces a value. The equivalent of the above code is the following:

```
/* condition */ ? /* Expression if condition is true */ : /* Expression i
f condition is false */
```

Because it uses three expressions, the `?:` operator is called the **ternary operator.**
The value that is produced by this operator is either the value of the truth expression or the value of the falsity expression. Because it is an expression, it

can be used at any place where values are needed.

The following examples contrast the `?:` operator to the `if-else` statement. The ternary operator is more concise and preferable to use in cases that are similar to these examples.

- **Initializating a value**
  To initialize a variable with 366 if it is leap year, 365 otherwise:

```
import std.stdio;

void main() {
    bool isLeapYear = false;

    int days = isLeapYear ? 366 : 365;
    writeln(days);
}
```

If we use an `if` statement, then one way to do this is to define the variable without an explicit initial value and then assign the intended value:

```
import std.stdio;

void main() {
    bool isLeapYear = false;

     int days;

    if (isLeapYear) {
        days = 366;

    } else {
        days = 365;
    }

    writeln(days);
}
```

An alternative is to initialize the variable with the non-leap year value and then increment it if it is a leap year:

```
import std.stdio;

void main() {
    bool isLeapYear = false;

    int days = 365;

    if (isLeapYear) {
        ++days;
    }

    writeln(days);
}
```

- **Displaying a message**
  This displays some part of a message differently depending on a condition:

```
import std.stdio;

void main() {
    bool isOptimistic = true;

    writeln("The glass is half ",isOptimistic ? "full." : "empty.");
}
```

With an `if` statement, the first and last parts of the message may be printed separately:

```
import std.stdio;

void main() {
    bool isOptimistic = true;

    write("The glass is half ");

    if (isOptimistic) {
        writeln("full.");

    } else {
        writeln("empty.");
    }
}
```

Alternatively, the entire message can be printed separately:

```
import std.stdio;

void main() {
    bool isOptimistic = true;

    if (isOptimistic) {
        writeln("The glass is half full.");

    } else {
        writeln("The glass is half empty.");
    }
}
```

- **Calculations**
  This demonstrates increasing the score of the winner in a backgammon game 2 points or 1 point depending on whether the game has ended with gammon:

```
import std.stdio;

void main() {
    int score;
    bool isGammon = false;

    score += isGammon ? 2:1;

    writeln(score);
}
```

Here is a straightforward equivalent using an `if` statement:

```
import std.stdio;

void main() {
    int score;
    bool isGammon = false;

    if (isGammon) {
      score += 2;

    } else {
        score += 1;
    }
    writeln(score);
}
```

An alternative also using an `if` is to first increment by one and then increment again if gammon:

```
import std.stdio;

void main() {
    int score;
    bool isGammon = false;

    ++score;

    if (isGammon) {
        ++score;
    }

    writeln(score);
}
```

As can be seen from the examples above, the code is more concise and clearer with the ternary operator in certain situations.

The type of the ternary expression

The value of the `?:` operator is either the value of the truth expression or the value of the falsity expression. The types of these two expressions need not be the same but they must have a common type.

The common type of two expressions is decided by a relatively complicated algorithm involving type conversions and inheritance. Additionally, depending on the expressions, the kind of result is either an `lvalue` or an `rvalue`. We will see these concepts in later chapters.

For now, accept common type as a type that can represent both of the values without requiring an explicit conversion. For example, the integer types `int` and `long` have a common type because they can both be represented as `long`. On the other hand, `int` and `string` do not have a common type because neither `int` nor `string` can automatically be converted to the other type.

Remember that a simple way of determining the type of an expression is using `typeof` and then printing its `.stringof` property:

```
int i;
double d;
auto result = someCondition ? i : d;
writeln(typeof(result).stringof);
```

Because `double` can represent `int` but not the other way around, the common type of the ternary expression above is `double`:

```
double
```

Example

As an example of two expressions that do not have a common type, let's look at composing a message that reports the number of items to be shipped. Let's print `A dozen` when the value equals 12: `A dozen items will be shipped.` Otherwise, let's have the message include the exact number: `3 items will be shipped.`
One might think that the varying part of the message can be selected with the `?:` operator:

```
import std.stdio;

void main() {
    int count = 11;
    // compilation error
    writeln((count == 12) ? "A dozen" : count, " items will be shipped.");

}
```

Unfortunately, the expressions do not have a common type because the type of `A dozen` is a `string` and the type of `count` is int.
A solution is to first convert `count` to string. The function `to!string` from the `std.conv` module produces a string value from the specified parameter:

```
import std.conv;
import std.stdio;

void main() {
    int count = 11;
    // compilation error
    writeln((count == 12) ? "A dozen" : to!string(count),
                            " items will be shipped.");
}
```

Now, becuase both of the selection expressions of the `?:` operator are of `string` type, the code compiles and prints the expected message.

In the next lesson, we will learn how to use lazy operators in D language.