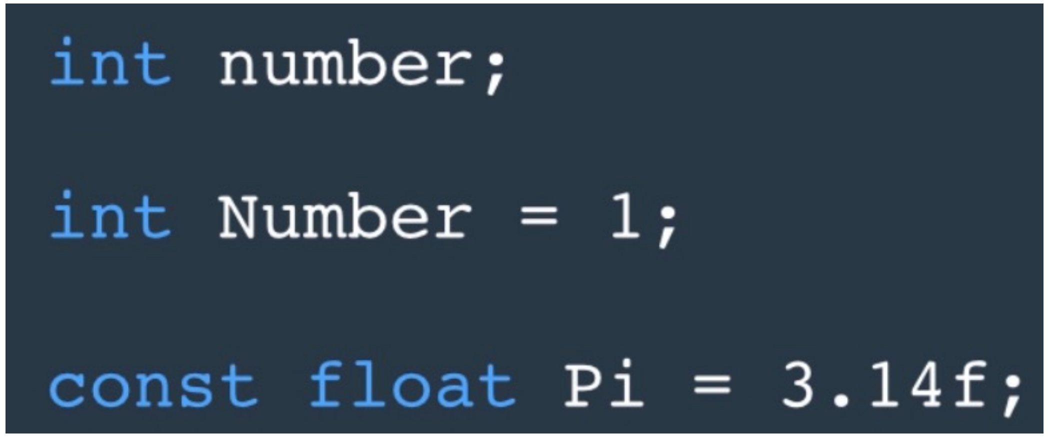
Chapter-2b

Primitive Types and Expressions

1. Variables
2. Constants
3. Scope
4. Overflow
5. Operators



```
int number;

int Number = 1;

const float Pi = 3.14f;
```

Variables – A name given to storage location in the memory
Constants – An immutable value

Data Type and Identifer is required to declare a variable followed by a semicolan. For constants. Its compulsory to assign a value to it.

Identifiers

1. Cannot starts with a number
   1. 1route – illegal
   2. oneroute – legal

2. No Whitespaces
   1. First Name – illegal
   2. firstName - legal

3. Cannot be a keyword
   1. int – illegal
   2. @int - legal

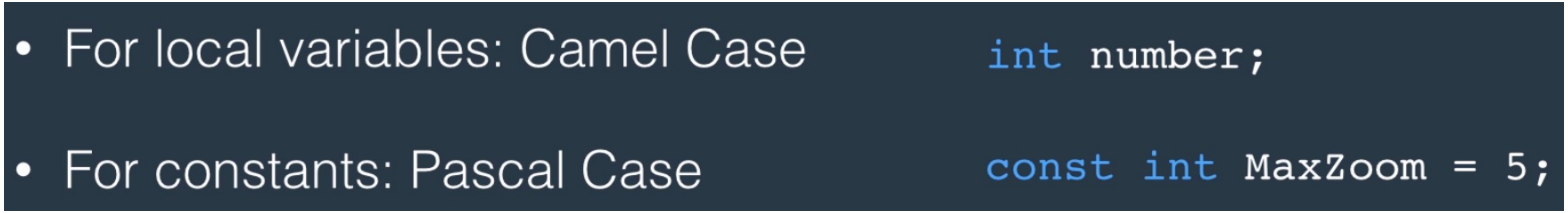
4. Always use meaningful names

Code need to be

1. Readable
2. Maintainable
3. Cleaner

Naming Conventions – C Language Family

Camel Case – firstName
Pascal Case – FirstName
Hungarian Notation – strFirstName (Came from C/C++ background. However, not liked by C# developers.



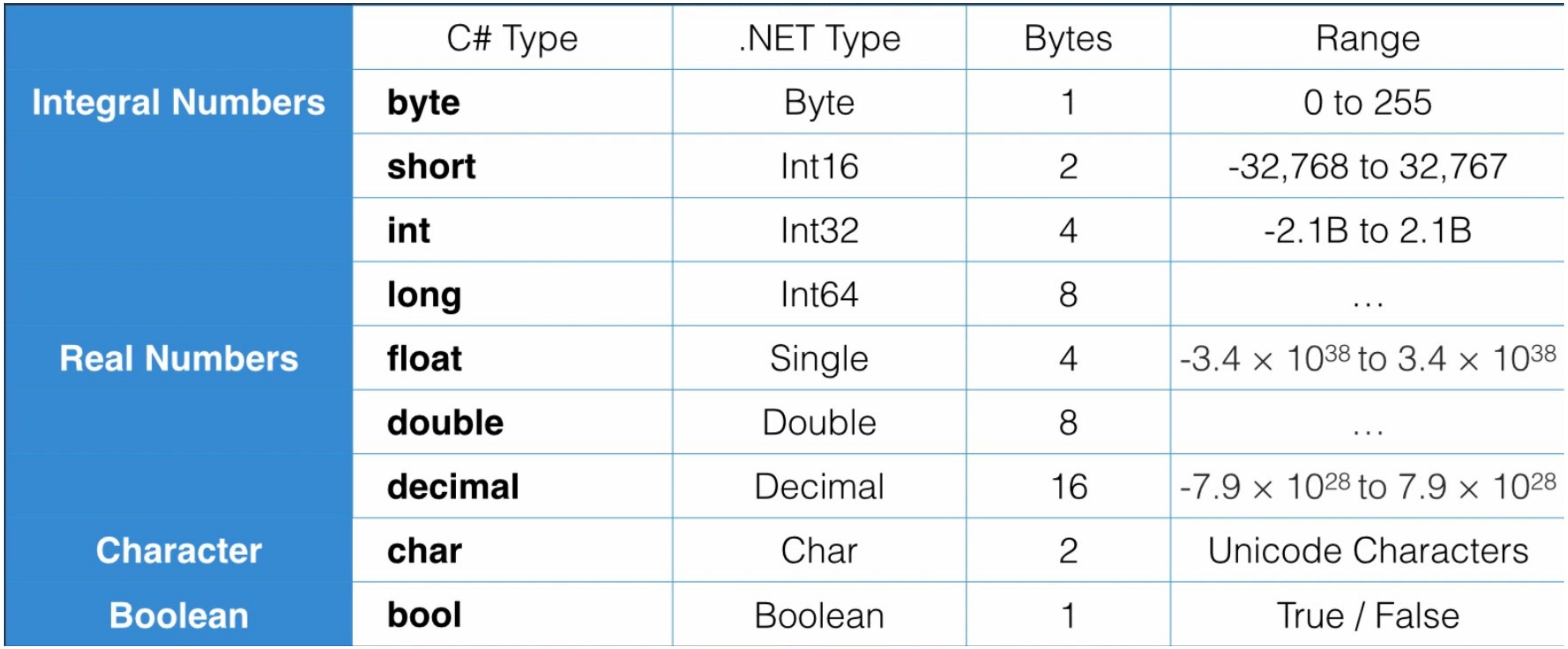
- For local variables: Camel Case

```
int number;
```

- For constants: Pascal Case

```
const int MaxZoom = 5;
```

Primitive Data Types



	C# Type	.NET Type	Bytes	Range
Integral Numbers	**byte**	Byte	1	0 to 255
	short	Int16	2	-32,768 to 32,767
	int	Int32	4	-2.1B to 2.1B
	long	Int64	8	…
Real Numbers	**float**	Single	4	-3.4×10^{38} to 3.4×10^{38}
	double	Double	8	…
	decimal	Decimal	16	-7.9×10^{28} to 7.9×10^{28}
Character	**char**	Char	2	Unicode Characters
Boolean	**bool**	Boolean	1	True / False

Real Numbers

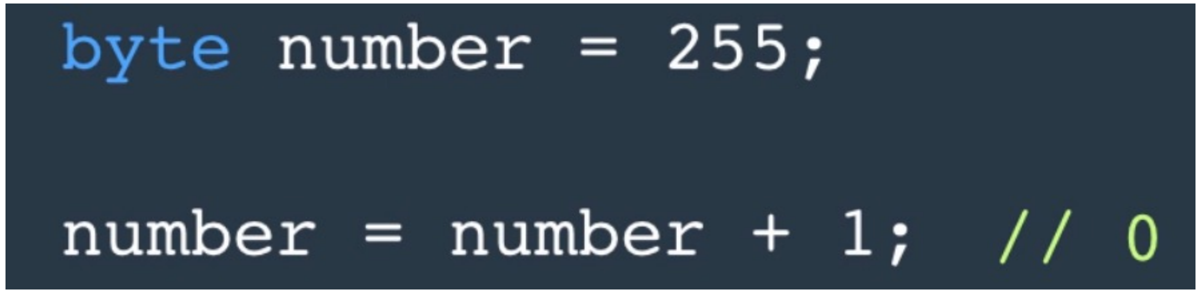
	C# Type	.NET Type	Bytes	Range
Real Numbers	**float**	Single	4	-3.4×10^{38} to 3.4×10^{38}
	double	Double	8	...
	decimal	Decimal	16	-7.9×10^{28} to 7.9×10^{28}

```
float number = 1.2f;

decimal number = 1.2m;
```

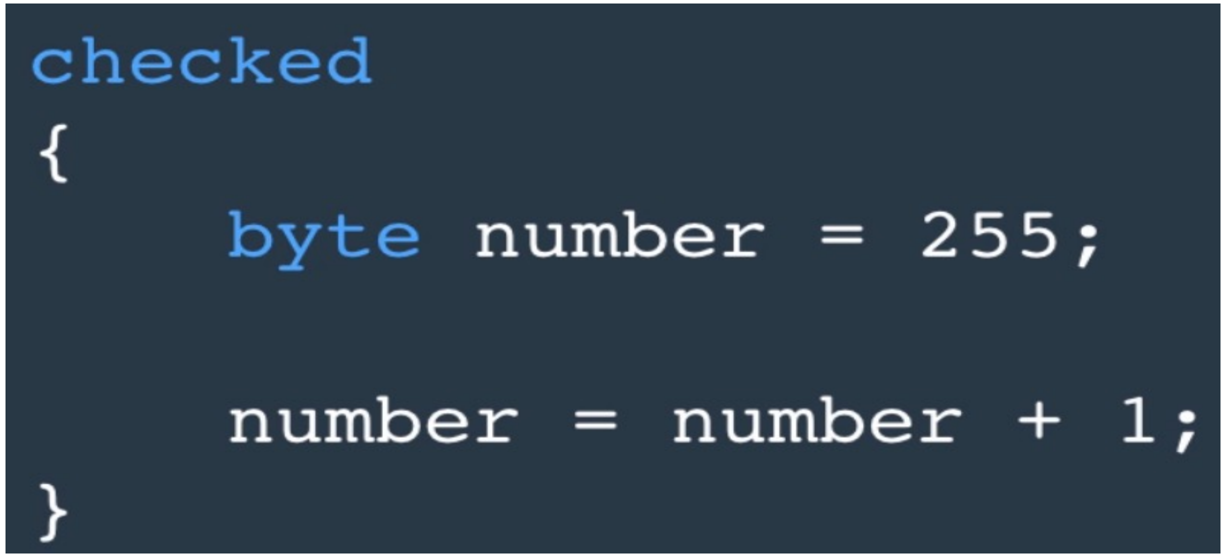
Non-Primitive Data Types

1. Strings
2. Arrays
3. Enum
4. Class

Overflowing



```
byte number = 255;

number = number + 1;  // 0
```



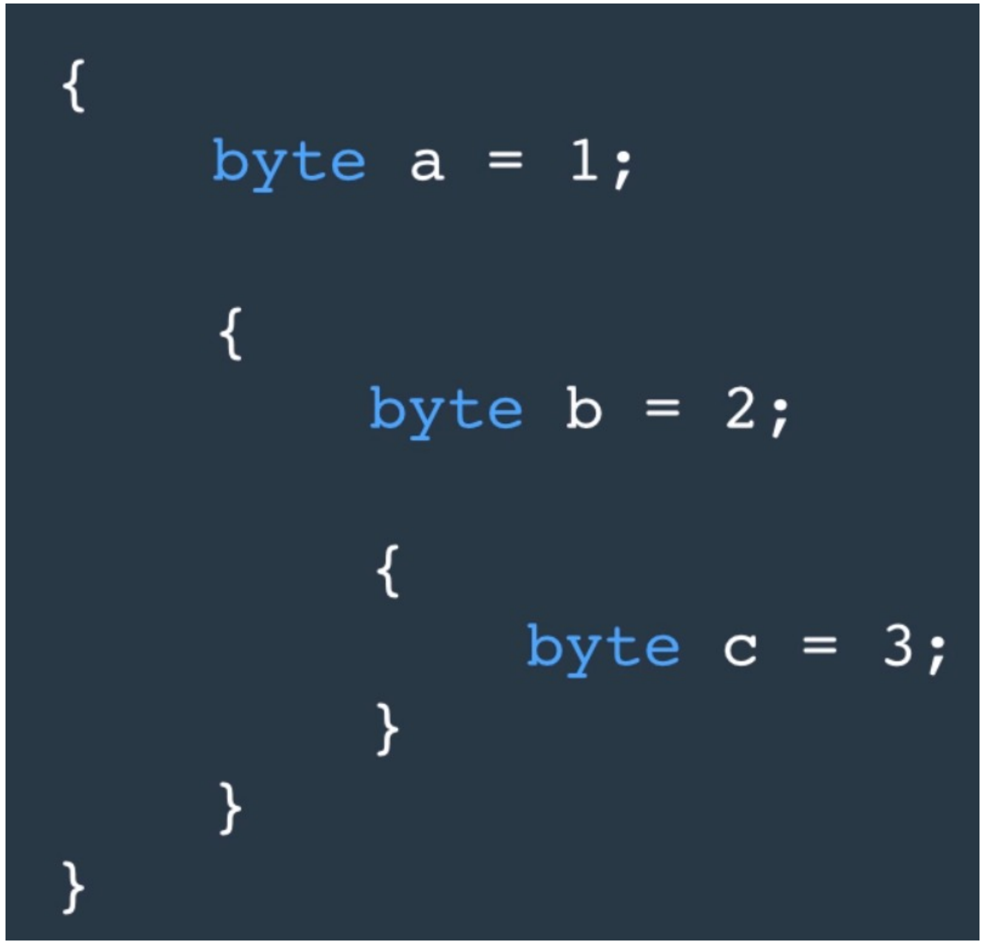
```
checked
{
     byte number = 255;

     number = number + 1;
}
```

Ariane 5 Explosion | A Very Costly Coding Error

https://www.youtube.com/watch?v=5tJPXYA0Nec

Scope

Scope – where a variable or constant have a meaning

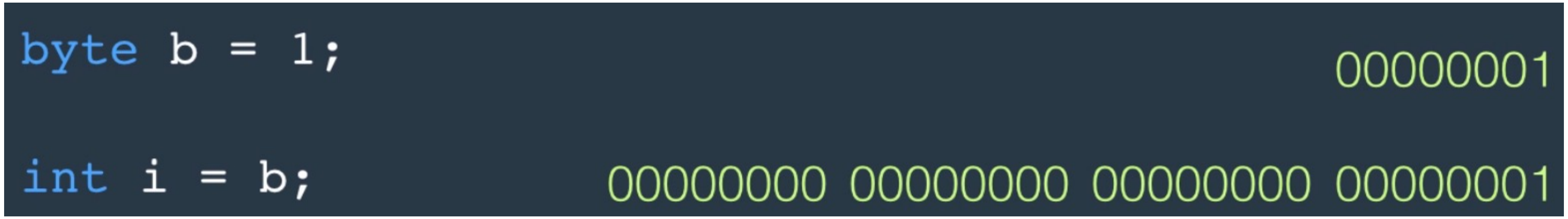


```
{
    byte a = 1;

    {
        byte b = 2;

        {
            byte c = 3;
        }
    }
}
```

Type Conversions

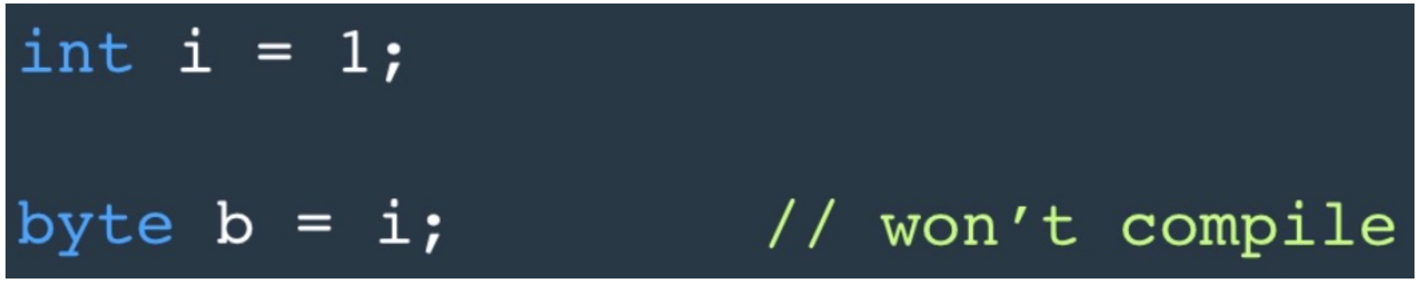
Implicit Type Conversion
Explicit Type Conversion (Casting)
Conversion between non compatible types

Implicit Type Conversion

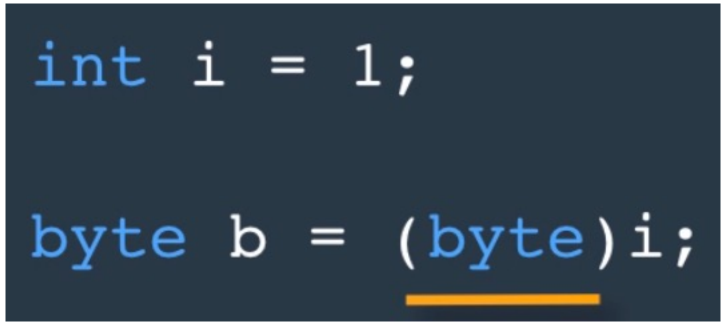


```
byte b = 1;                                                00000001

int i = b;                 00000000 00000000 00000000 00000001
```

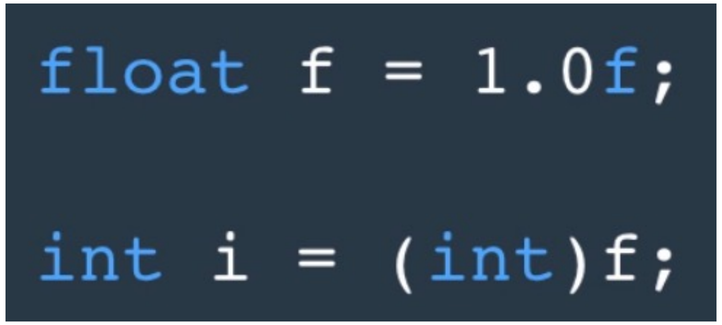
Explicit Types Conversion



```
int i = 1;

byte b = i;          // won't compile
```

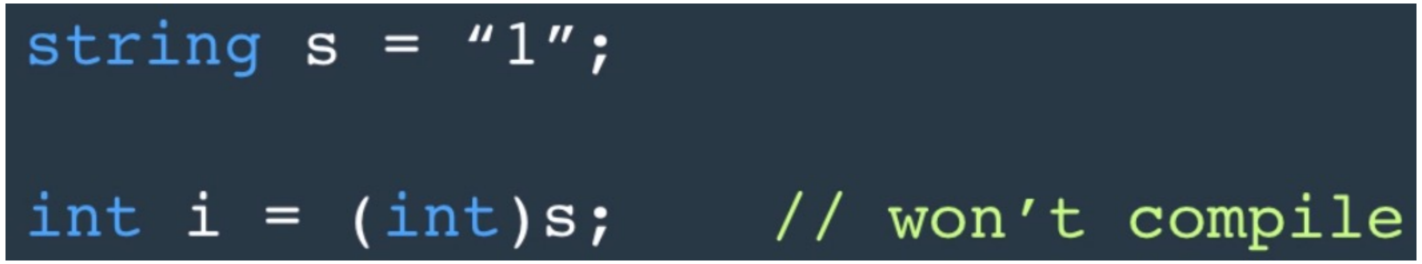


```
int i = 1;

byte b = (byte)i;
```



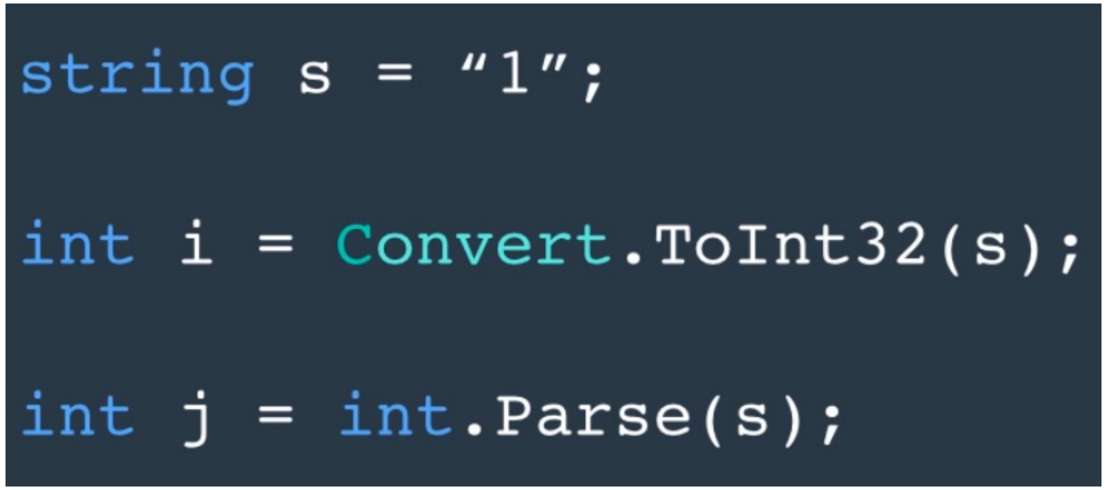
```
float f = 1.0f;

int i = (int)f;
```

Non-Compatible Types



```
string s = “1”;

int i = (int)s;     // won’t compile
```

Use Convert class defined in System Namespace or Parse method

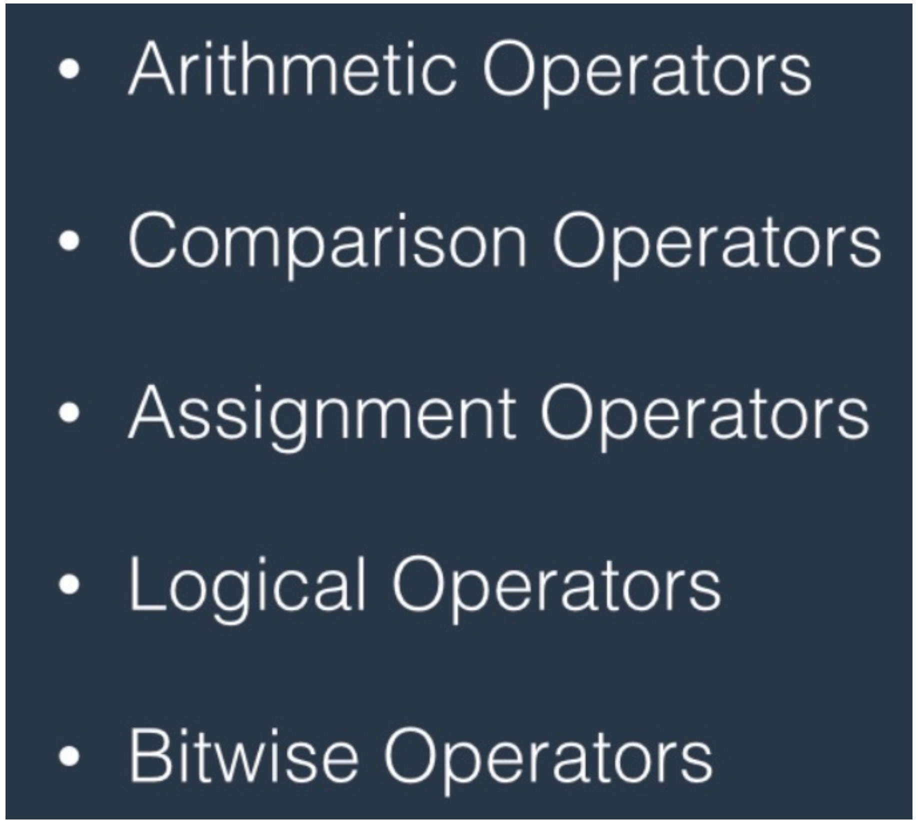


```
string s = “1”;

int i = Convert.ToInt32(s);

int j = int.Parse(s);
```

Convert Class

- ToByte()
- ToInst16()
- ToInt32()
- ToInt64()

C# Operators

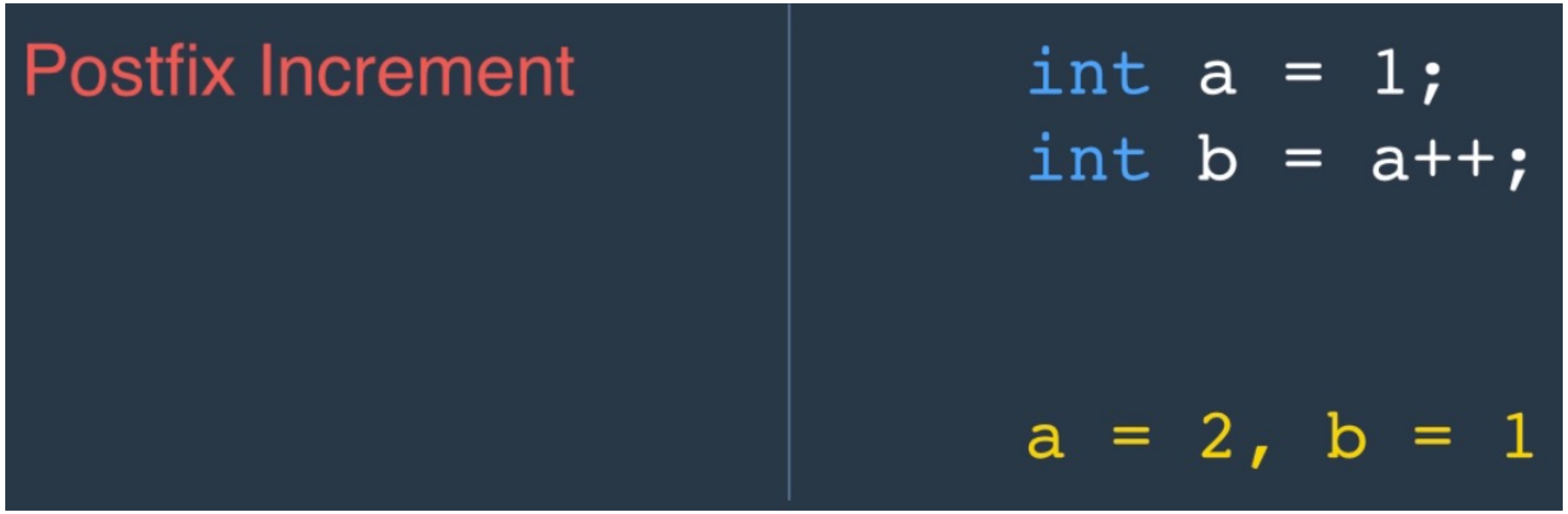


- Arithmetic Operators
- Comparison Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators

Arithmetic Operators

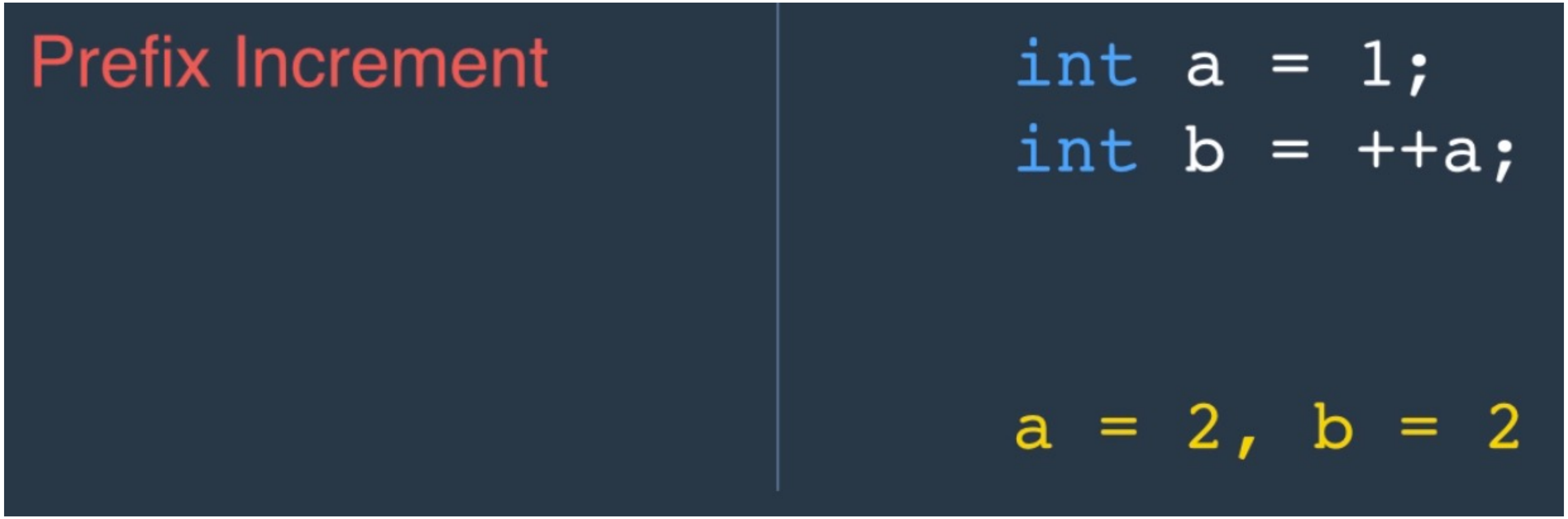
	Operator	Example
Add	**+**	a + b
Subtract	**-**	a - b
Multiply	*****	a * b
Divide	**/**	a / b
Remainder	**%**	a % b

	Operator	Example	Same as
Increment	**++**	a++	a = a + 1
Decrement	**--**	a--	a = a - 1



Postfix Increment

```
int a = 1;
int b = a++;


a = 2, b = 1
```



Prefix Increment

```
int a = 1;
int b = ++a;
```

```
a = 2, b = 2
```

Comparison Operators

	Operator	Example
Equal	**==**	a == b
Not Equal	**!=**	a != b
Greater than	**>**	a > b
Greater than or equal to	**>=**	a >= b
Less than	**<**	a < b
Less than or equal to	**<=**	a <= b

Assignment Operators

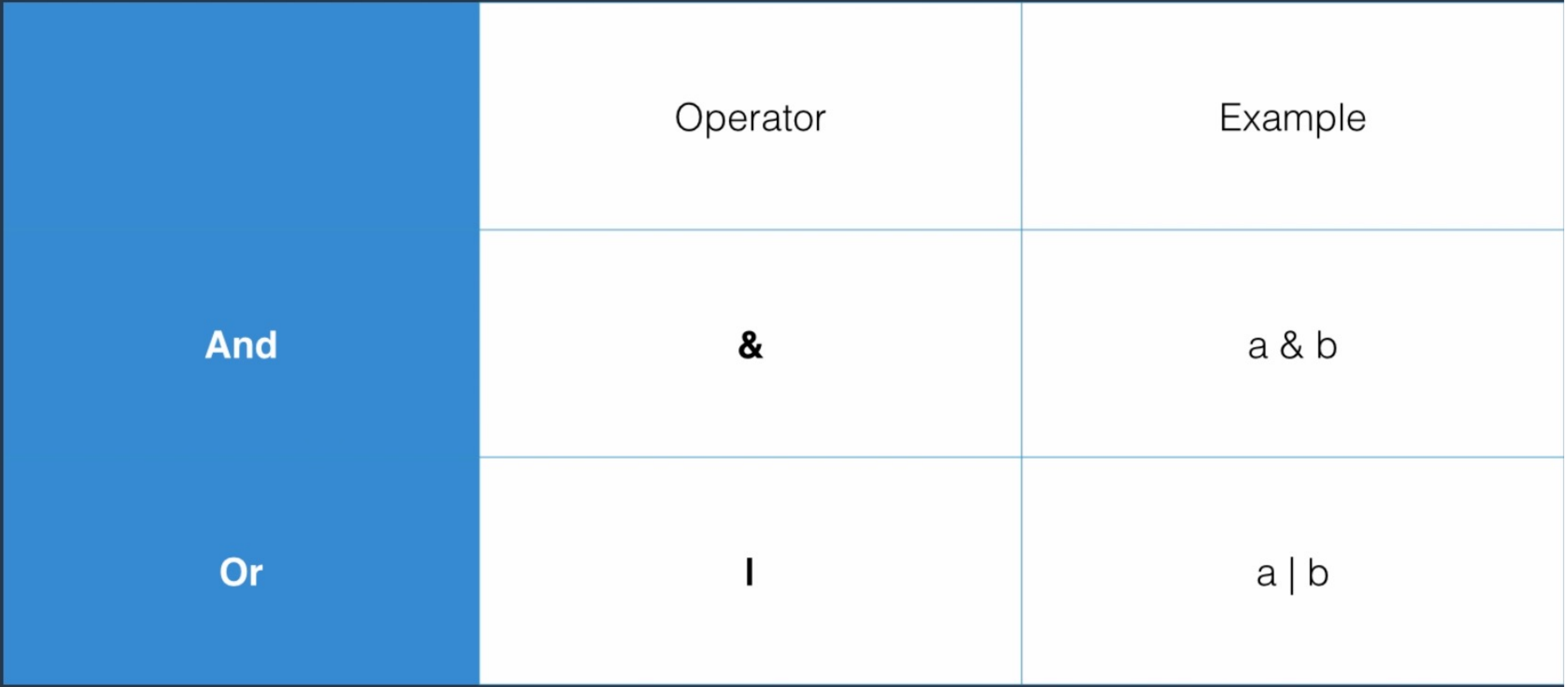
	Operator	Example	Same as
Assignment	**=**	a = 1	
Addition assignment	**+=**	a += 3	a = a + 3
Subtraction assignment	**-=**	a -= 3	
Multiplication assignment	***=**	a *= 3	
Division assignment	**/=**	a /= 3	

Logical Operators

	Operator	Example
And	**&&**	a && b
Or	**||**	a || b
Not	**!**	!a

Bitwise Operators

Used in low level programming, sockets, encryption,



	Operator	Example
And	**&**	a & b
Or	**|**	a | b

Primitive Types and Expressions

Follow Book

Primitive Types and Expressions

Follow Book