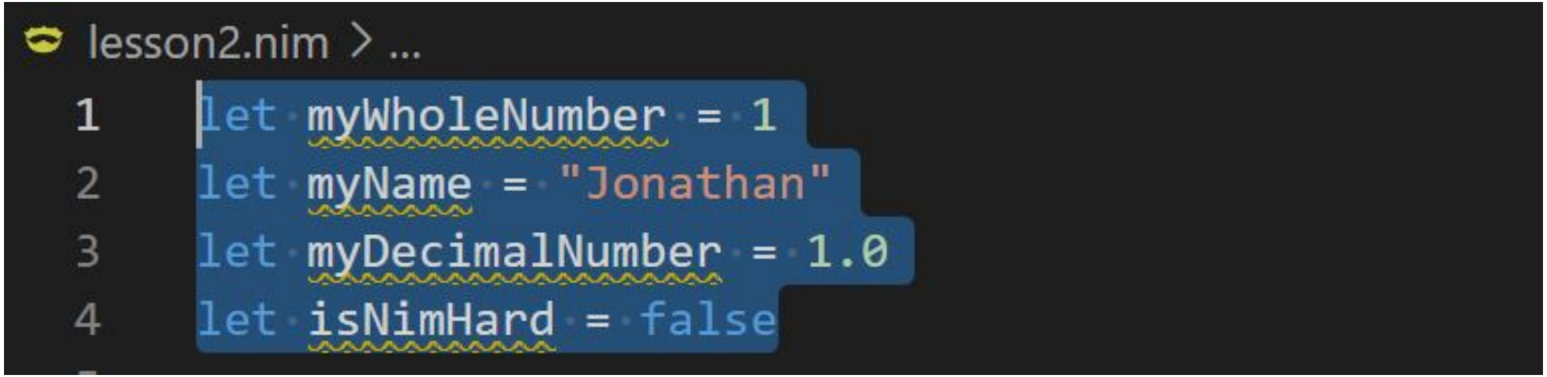
Defining Variables: Basics & Storage Keywords

STORAGE KEYWORD VARIABLE NAME = VALUE



```nim
let myWholeNumber = 1
let myName = "Jonathan"
let myDecimalNumber = 1.0
let isNimHard = false
```

Defining Variables: Basics & Storage Keywords

STORAGE KEYWORD	VARIABLE NAME	=	VALUE
let var const	Mostly free to choose - some conventions and notes about case sensitivity and insensitivity		Literal values - 1, 1.1, “c” , “string” Returned value from a function / procedure sqrt(2)

Storage Keywords

Mutable Variable - Can change the value it holds from its initial value, e.g. age, height, weight

Immutable Variable - cannot change value, e.g. mathematical and physical constants, pi = 3.14.... , e= 2.718...

----

let - indicates a variable is immutable. Must be initialised when defined!

var - a mutable variable. Does not have to be initialised.

const - similar to **let**, but value must be known at compile time

let

```
let x = 12 # OK, x is initialised to 12

x = 13  # Error: x cannot be assigned to
```

var

var x= 12 # ok, initialised as 12

x = 14 # okay, can reassign

const

```
var x =1

let y = x+1 # Okay

const z = x+1 # error
```

Similar to let, but evaluated at compile time. Let is evaluated at run time. We will see this situationally as we go along when applying nim to real code.

Multiple definition

```
let:
        c = 299,792,458
        pi = 3.14159265359

var:
        x = 0
        y = 0.1
        vx = 1
        vy = 2
        ax = 0
        ay = 0
```

NB - , in numbers are ignored.
Readability feature in nim that isn’t very common in other languages

NB: Indentation defines blocks and scope in nim (as opposed to {}) , we will see this more soon

Variable Names

Partial Case Insensitivity

The variable identifier / name is case insensitive *except for the first letter*.

The first letter is case sensitive to help with a convention - variables start with lowercase letter, and user-defined types start with an uppercase letter.

Style Insensitivity

Underscores are ignored in identifiers. In combination with the partial case insensitivity gives style insensitivity

myNumber is equivalent to my_number

Allows to write e.g. a lib in one style, but have a user use it in another if they prever.

Therefore camelCase and snake_case supported. Nim community tries to get all to use camelCase.

Built-in data types in Nim

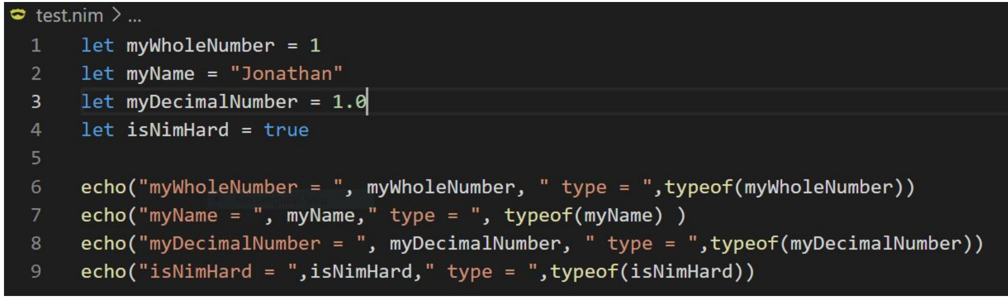
- A data type is an attribute of data which tells the compiler how the programmer intends to use the data.
- Every variable must have a **Type**
- So far, we have defined & initialised variables where the type is inferred for us by Nim
- We must understand the different types and sometimes we want to, or must, explicitly declare them.

Type inference example

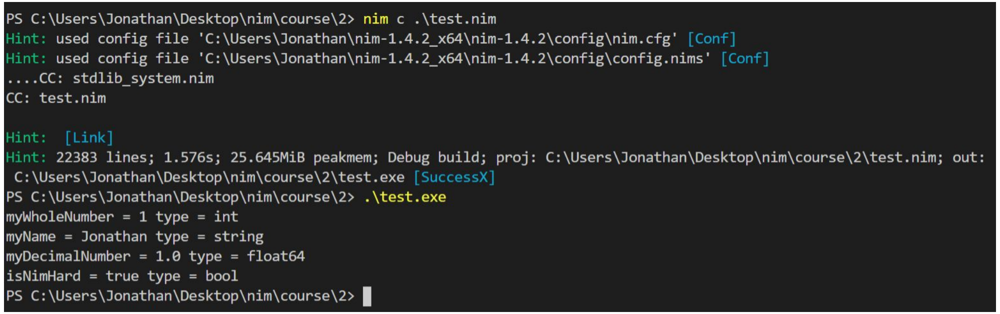
- Can use the **typeof** procedure to determine the type of a variable or an expression

- Can call this in our **echo** statements to interrogate the type of the variables the compiler has inferred for us

- Decides based on the form of the literal expressions



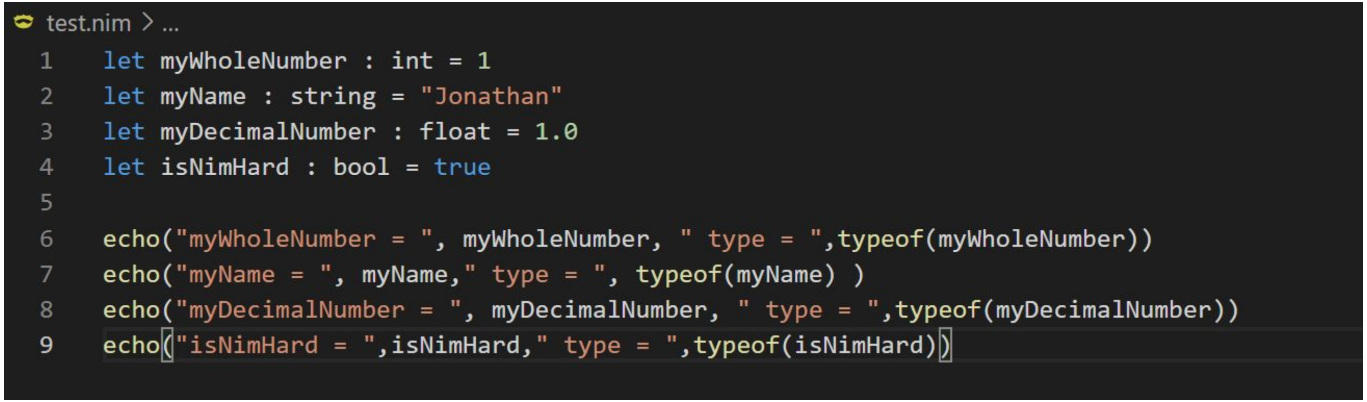
```nim
let myWholeNumber = 1
let myName = "Jonathan"
let myDecimalNumber = 1.0
let isNimHard = true

echo("myWholeNumber = ", myWholeNumber, " type = ",typeof(myWholeNumber))
echo("myName = ", myName," type = ", typeof(myName) )
echo("myDecimalNumber = ", myDecimalNumber, " type = ",typeof(myDecimalNumber))
echo("isNimHard = ",isNimHard," type = ",typeof(isNimHard))
```



```
PS C:\Users\Jonathan\Desktop\nim\course\2> nim c .\test.nim
Hint: used config file 'C:\Users\Jonathan\nim-1.4.2_x64\nim-1.4.2\config\nim.cfg' [Conf]
Hint: used config file 'C:\Users\Jonathan\nim-1.4.2_x64\nim-1.4.2\config\config.nims' [Conf]
....CC: stdlib_system.nim
CC: test.nim

Hint:  [Link]
Hint: 22383 lines; 1.576s; 25.645MiB peakmem; Debug build; proj: C:\Users\Jonathan\Desktop\nim\course\2\test.nim; out:
 C:\Users\Jonathan\Desktop\nim\course\2\test.exe [SuccessX]
PS C:\Users\Jonathan\Desktop\nim\course\2> .\test.exe
myWholeNumber = 1 type = int
myName = Jonathan type = string
myDecimalNumber = 1.0 type = float64
isNimHard = true type = bool
PS C:\Users\Jonathan\Desktop\nim\course\2>
```

Explicit type declaration

STORAGE KEYWORD VARIABLE NAME : TYPE = VALUE



test.nim > ...

```nim
let myWholeNumber : int = 1
let myName : string = "Jonathan"
let myDecimalNumber : float = 1.0
let isNimHard : bool = true

echo("myWholeNumber = ", myWholeNumber, " type = ",typeof(myWholeNumber))
echo("myName = ", myName," type = ", typeof(myName) )
echo("myDecimalNumber = ", myDecimalNumber, " type = ",typeof(myDecimalNumber))
echo("isNimHard = ",isNimHard," type = ",typeof(isNimHard))
```

Built-in data types in Nim

Basic built in types are:

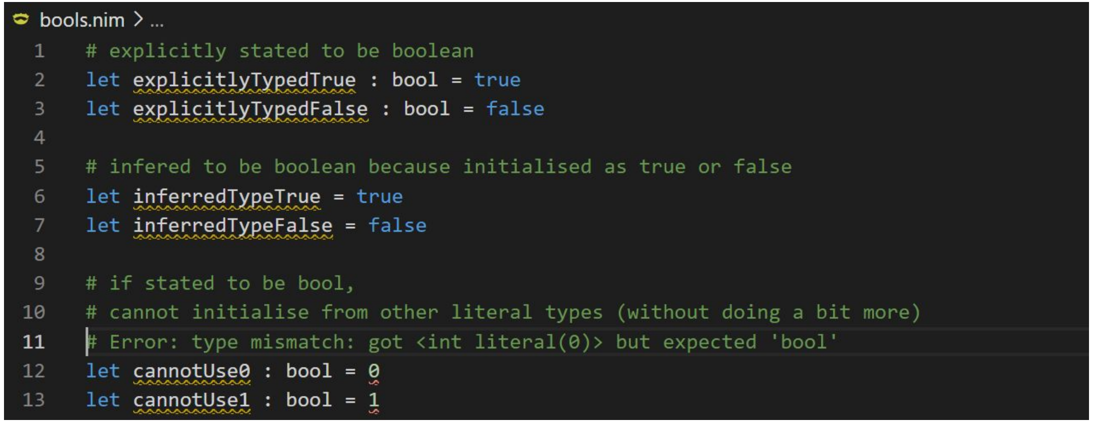
- Booleans - **bool** - can be true or false
- Characters - **char** - holds single ASCII character (1 byte long)
- Strings - **string** - holds character sequences
- Integers - **int** - byte size dependent on compiler / platform - specific widths exist e.g. **int64**
- Floating point (decimal) number - **float** - always 64 bit, other widths must be accessed specifically e.g. **float32**

Notice names begin lower case - this is for built in only. User-defined types should begin with caps, e.g. MyType

Built-in data types in Nim

Booleans / **bool**

- Either **true** or **false**
- The operators **not**, **and**, **or**, **xor**, **<**, **<=**, **>**, **>=**, **!=**, **==** are defined for the bool type.
- Often used in conditional logic, will explore this type in detail in that section.



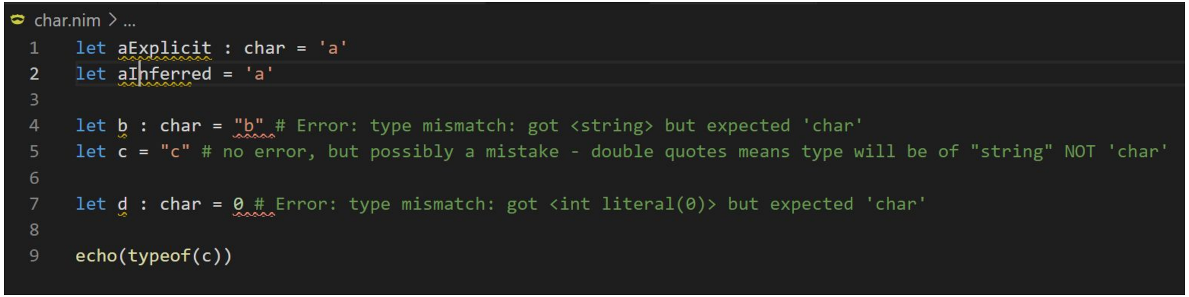
bools.nim > ...

```nim
# explicitly stated to be boolean
let explicitlyTypedTrue : bool = true
let explicitlyTypedFalse : bool = false

# infered to be boolean because initialised as true or false
let inferredTypeTrue = true
let inferredTypeFalse = false

# if stated to be bool,
# cannot initialise from other literal types (without doing a bit more)
# Error: type mismatch: got <int literal(0)> but expected 'bool'
let cannotUse0 : bool = 0
let cannotUse1 : bool = 1
```

Built-in data types in Nim

Characters / **char**

- Represents a single “symbol”, “letter” or “glyph” - a **character**
- 1 byte long so can hold ASCII chars (but can only partially represent UTF-8 chars don’t worry about this)
- Character literals are enclosed with single quotes ‘’ (as opposed to double quotes “”, for string literals)



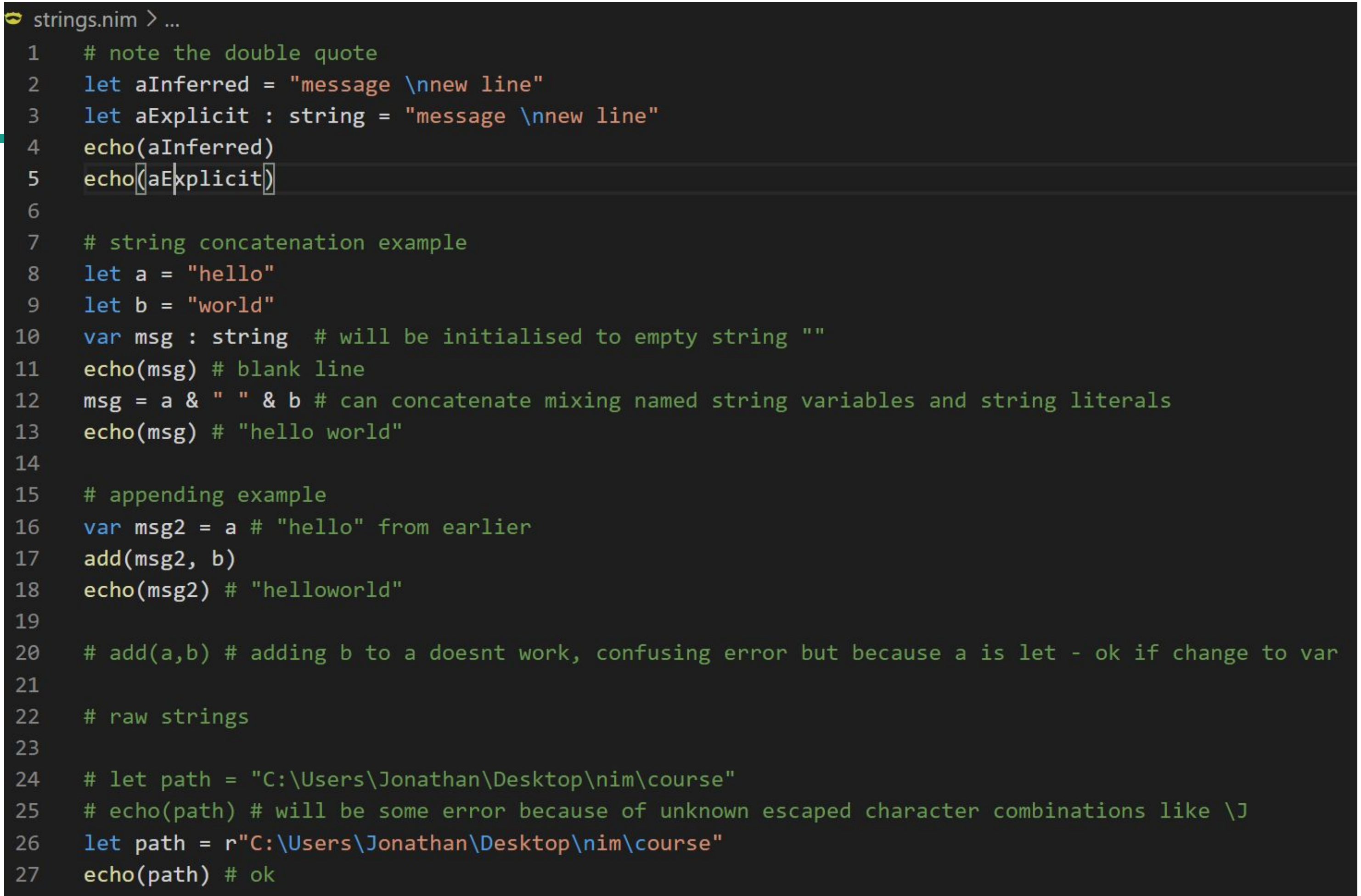
char.nim > ...

```
let aExplicit : char = 'a'
let aInferred = 'a'

let b : char = "b" # Error: type mismatch: got <string> but expected 'char'
let c = "c" # no error, but possibly a mistake - double quotes means type will be of "string" NOT 'char'

let d : char = 0 # Error: type mismatch: got <int literal(0)> but expected 'char'

echo(typeof(c))
```

Built-in data types in Nim

Strings / **string**

- Represents a series or collection of characters.
- Use double quotes for literal
- \ escapes for special characters, e.g. \n [newline]
- Prefix with r for a raw string literal, where \ are not escapes (e.g. windows filepaths)
- Strings can be concatenated with the & operator
- Strings can be added with the add() procedure

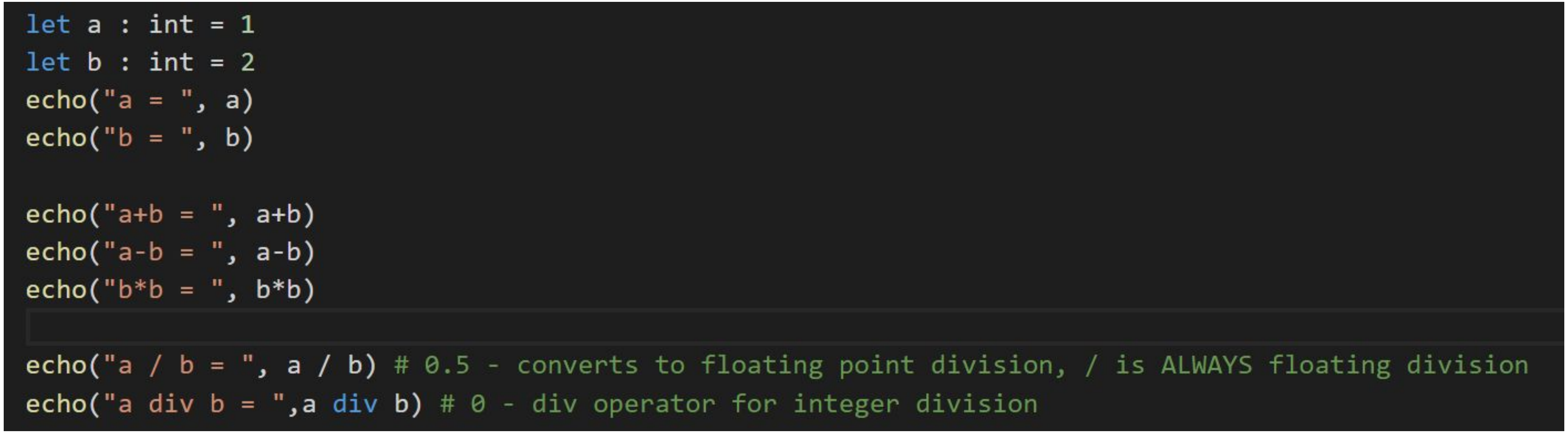


```nim
# note the double quote
let aInferred = "message \nnew line"
let aExplicit : string = "message \nnew line"
echo(aInferred)
echo(aExplicit)

# string concatenation example
let a = "hello"
let b = "world"
var msg : string  # will be initialised to empty string ""
echo(msg) # blank line
msg = a & " " & b # can concatenate mixing named string variables and string literals
echo(msg) # "hello world"

# appending example
var msg2 = a # "hello" from earlier
add(msg2, b)
echo(msg2) # "helloworld"

# add(a,b) # adding b to a doesnt work, confusing error but because a is let - ok if change to var

# raw strings

# let path = "C:\Users\Jonathan\Desktop\nim\course"
# echo(path) # will be some error because of unknown escaped character combinations like \J
let path = r"C:\Users\Jonathan\Desktop\nim\course"
echo(path) # ok
```

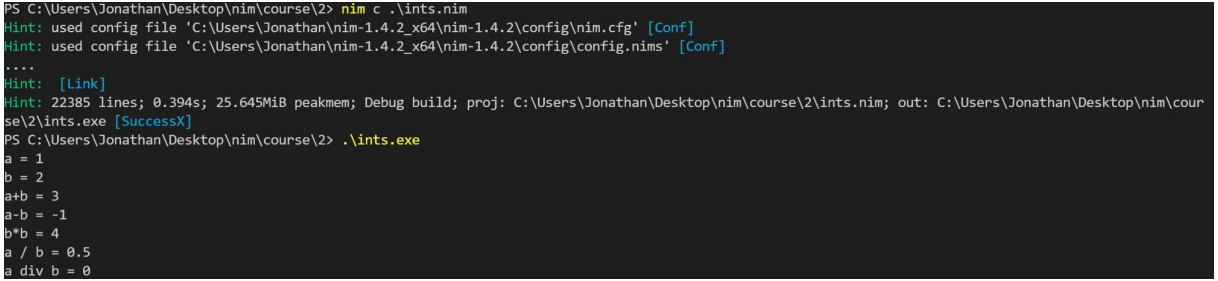
Built-in data types in Nim

Integers / **int**

- This type holds **whole numbers**, both +ve and -ve.
- Common operators defined `+ - * div mod < <= == != > >=` so we can do maths and other useful things with them
- Bit width (tf maximum values) platform dependent, usually 64 bit on a modern desktop. Have types that have fixed-widths e.g. int8, int16, int32, int64.
- Use this type as a counter



```
let a : int = 1
let b : int = 2
echo("a = ", a)
echo("b = ", b)

echo("a+b = ", a+b)
echo("a-b = ", a-b)
echo("b*b = ", b*b)

echo("a / b = ", a / b) # 0.5 - converts to floating point division, / is ALWAYS floating division
echo("a div b = ",a div b) # 0 - div operator for integer division
```

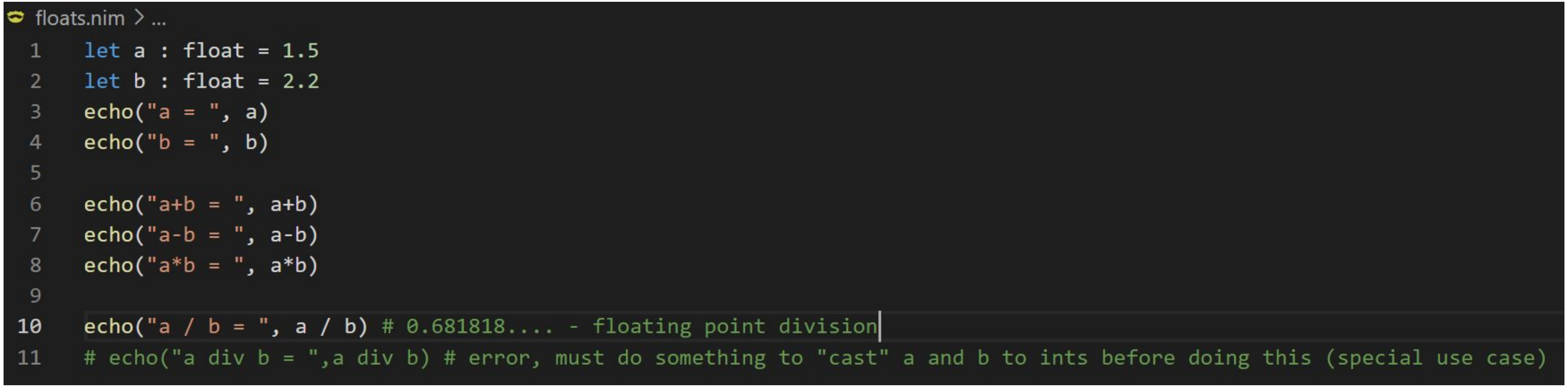


```
PS C:\Users\Jonathan\Desktop\nim\course\2> nim c .\ints.nim
Hint: used config file 'C:\Users\Jonathan\nim-1.4.2_x64\nim-1.4.2\config\nim.cfg' [Conf]
Hint: used config file 'C:\Users\Jonathan\nim-1.4.2_x64\nim-1.4.2\config\config.nims' [Conf]
....
Hint:  [Link]
Hint: 22385 lines; 0.394s; 25.645MiB peakmem; Debug build; proj: C:\Users\Jonathan\Desktop\nim\course\2\ints.nim; out: C:\Users\Jonathan\Desktop\nim\cour
se\2\ints.exe [SuccessX]
PS C:\Users\Jonathan\Desktop\nim\course\2> .\ints.exe
a = 1
b = 2
a+b = 3
a-b = -1
b*b = 4
a / b = 0.5
a div b = 0
```

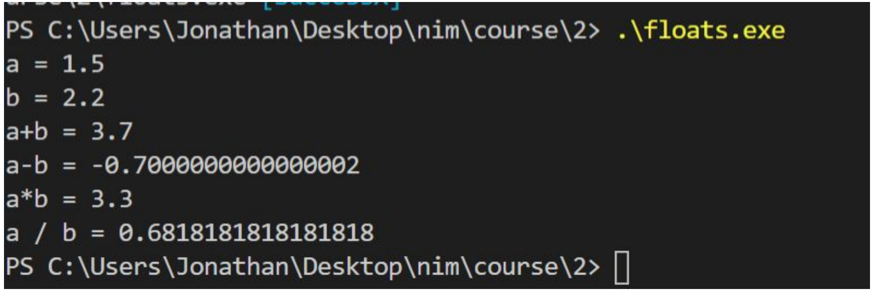
Built-in data types in Nim

Floating point number / **float**

- This type holds numbers with fractional values (i.e. decimals)
- Common operators defined so can do maths and other useful things `+ - * / < <= == != > >=`
- Currently bit width of **float** is always 64 bits. **float32** is an alternative.
- Use this type for real numbers



```nim
let a : float = 1.5
let b : float = 2.2
echo("a = ", a)
echo("b = ", b)

echo("a+b = ", a+b)
echo("a-b = ", a-b)
echo("a*b = ", a*b)

echo("a / b = ", a / b) # 0.681818.... - floating point division
# echo("a div b = ",a div b) # error, must do something to "cast" a and b to ints before doing this (special use case)
```



```
PS C:\Users\Jonathan\Desktop\nim\course\2> .\floats.exe
a = 1.5
b = 2.2
a+b = 3.7
a-b = -0.7000000000000002
a*b = 3.3
a / b = 0.6818181818181818
PS C:\Users\Jonathan\Desktop\nim\course\2>
```