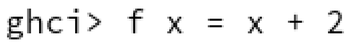
Definitons:

Parameter:

- A parameter represents a value, that will be passed to a function when that function is called.

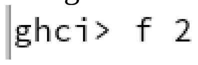


```
ghci> f x = x + 2
```

X is the parameter

Argument:

- An argument is an input value the function is applied to.

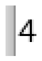


```
ghci> f 2
```

2 is the argument.

Value:

- A value is an expression that cannot be reduced or evaluated any further.

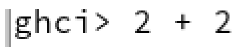


```
4
```

4 is a Value.

Expression:

- An expression is a combination of symbols that conforms to syntactic rules and can be reduced to some result.

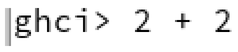


```
ghci> 2 + 2
```

2 + 2 is an expression.

Infix notation:

- Infix notation is the style used in arithmetic and logic. Infix means that the operator is placed between the operands or arguments.
- Operators are functions that are infix by default.

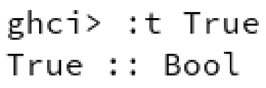


```
ghci> 2 + 2
```

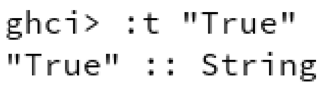
+ is an Infix notation

Type Datatype:

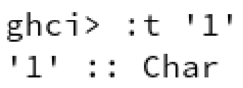
- A type or datatype is a classification of values or data. Types in Haskell determine what values are members of the type.



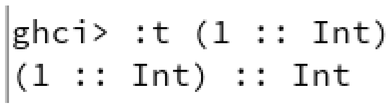
```
ghci> :t True
True :: Bool
```



```
ghci> :t "True"
"True" :: String
```



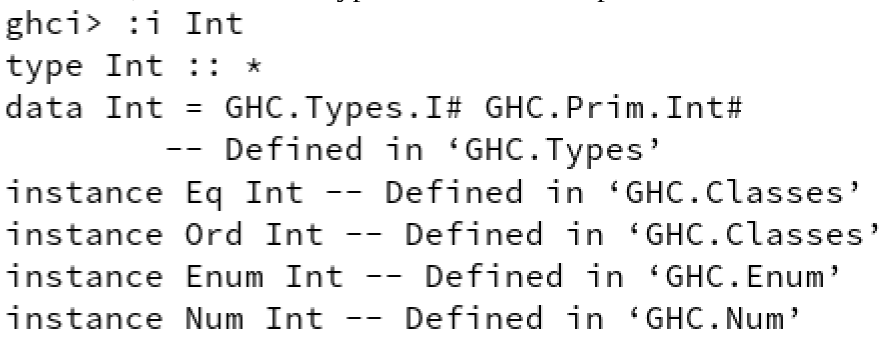
```
ghci> :t '1'
'1' :: Char
```



```
ghci> :t (1 :: Int)
(1 :: Int) :: Int
```

Type class:

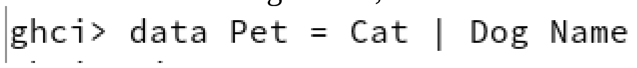
- A type class is a set of operations defined to a type. When a type has an instance of a type class, values of that type can be used in operations defined for that type class.



```
ghci> :i Int
type Int :: *
data Int = GHC.Types.I# GHC.Prim.Int#
        -- Defined in ‘GHC.Types’
instance Eq Int -- Defined in ‘GHC.Classes’
instance Ord Int -- Defined in ‘GHC.Classes’
instance Enum Int -- Defined in ‘GHC.Enum’
instance Num Int -- Defined in ‘GHC.Num’
```

Data constructors:

- Data constructors in Haskell provide a means of creating values that inhabit a given type. Data constructors in Haskell have a type and can either be constant values (nullary) or take one or more arguments, like functions.

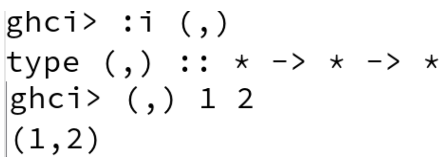


```
ghci> data Pet = Cat | Dog Name
```

Cat is a nullary data constructor and Dog is a data constructor that takes an argument:

Type constructor:

- Type constructors in Haskell are not values and can only be used in type signatures. Just as data declarations generate data constructors in order to create the values that inhabit a given type, data declarations also generate type constructors, which can be used to denote that type. In the above example, Pet is the type constructor. The tuple constructor needs to be applied to values in order a Expression.



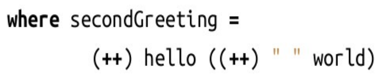
```
ghci> :i (,)
type (,) :: * -> * -> *
ghci> (,) 1 2
(1,2)
```

Scope:

- A scope is where a variable can be validly referred to by name in a program. Something of global scope can be referred to anywhere in the entire program itself. A declaration at the top level of a Haskell module has module scope

Local bindings:

- The where/let clause creates local bindings for expressions that are not in scope at the top level.



```
where secondGreeting =
        (++) hello ((++) " " world)
```

Top level bindings:

- Top level bindings in Haskell are bindings that stand outside of any other declaration. The main feature of top-level bindings is that they can be made available to other modules within your programs