# Tupled Function



## Tupled Function

With functions bounded to arities up to 22 it was possible to generalize some operation on all function types using overloading. Now that we have functions and tuples generalized to arities above 22 overloading is not an option anymore. The type class `TupleFunction` provides a way to abstract directly over a function of any arity converting it to an equivalent function that receives all arguments in a single tuple.

```
/** Type class relating a `FunctionN[..., R]` with an equivalent tupled function `Function1[TupleN[...], R]`  *
 *  @tparam F a function type
 *  @tparam G a tupled function type (function of arity 1 receiving a tuple as
 */
@implicitNotFound("${F} cannot be tupled as ${G}")
sealed trait TupledFunction[F, G] {
  def tupled(f: F): G
  def untupled(g: G): F
}
```

The compiler will synthesize an instance of `TupledFunction[F, G]` if:

- `F` is a function type of arity `N`
- `G` is a function with a single tuple argument of size `N` and its types are equal to the arguments of `F`
- The return type of `F` is equal to the return type of `G`
- `F` and `G` are the same sort of function (both are `( ... ) ⇒ R` or both are `( ... ) ?=> R`)
- If only one of `F` or `G` is instantiated the second one is inferred.

## Examples

`TupledFunction` can be used to generalize the `Function1.tupled`, ... `Function22.tupled` methods to functions of any arities. The following defines `tupled` as extension method (full example).

```
/** Creates a tupled version of this function: instead of N arguments,  *  it a
 *
 *  @tparam F the function type
 *  @tparam Args the tuple type with the same types as the function arguments
 *  @tparam R the return type of F
 */
extension [F, Args <: Tuple, R](f: F)
  def tupled(using tf: TupledFunction[F, Args => R]): Args => R = tf.tupled(f)
```

`TupledFunction` can be used to generalize the `Function.untupled` to a function of any arities (full example)

```
/** Creates an untupled version of this function: instead of a single
argument of type [[scala.Tuple]] with N elements,  *  it accepts N arguments.
 *
 *  This is a generalization of [[scala.Function.untupled]] that work on functi
 *
 *  @tparam F the function type
 *  @tparam Args the tuple type with the same types as the function arguments
 *  @tparam R the return type of F
 */
extension [F, Args <: Tuple, R](f: Args => R)
  def untupled(using tf: TupledFunction[F, Args => R]): F = tf.untupled(f)
```

`TupledFunction` can also be used to generalize the `Tuple1.compose` and `Tuple1.andThen` methods to compose functions of larger arities and with functions that return tuples.

```
/** Composes two instances of TupledFunction into a new TupledFunction, with
this function applied last.  *
 *  @tparam F a function type
 *  @tparam G a function type
 *  @tparam FArgs the tuple type with the same types as the function arguments
 *  @tparam GArgs the tuple type with the same types as the function arguments
 *  @tparam R the return type of F
 */
extension [F, G, FArgs <: Tuple, GArgs <: Tuple, R](f: F)
  def compose(g: G)(using tg: TupledFunction[G, GArgs => FArgs], tf: TupledFunc
  (x: GArgs) => tf.tupled(f)(tg.tupled(g)(x))
}
```

Captur... Scala 3 ...