Only one implicit is tried.

`x + y` will never be rewritten.

```
convert1(convert2(x)) + y
```

Circumvent this with implicits take implicit parameters.

Whenever code type checks as it is written, no implicits are attempted.

Corollary: you can always replace implicit identifiers with explicit ones to make code longer but clearer.

Only definitions marked implicit are available.

An implicit conversion is only inserted if there is no other possible conversion to insert.

There is no "best match" rule.

Use when implicit parameter is never explicitly used in method body, but only present to introduce implicit conversion ability to body.

```
def maxList[T] (elements: List[T])
  (implicit orderer: T => Ordered[T]):T
def maxList[T <% Ordered[T]]
  (elements: List[T]):T
```

- Explicit invocation.
- Selectively importing one conversion method but not another.

Modular reasoning. You only need to consider code that is imported or explicitly referenced through a fully qualified name.

They provide information about a type mentioned explicitly in the earlier parameter list.

Default arguments are controlled at the point of definition.

Implicit parameters are chosen at the point of use and, as such, cannot be called in short form without the implicit being defined.

- Try using conversions explicitly to get a new error message or to establish scoping problem.

- Use `-Xprint:typer` option to see your code after its implicits have been expanded.

- Conversion to an expected type (e.g. to match a parameter).
- Conversions of the receiver of a selection (e.g., "abc".exists(...)`).
- Implicit parameters.

```
Preamble object.
import Preamble._
```

Any variable, function, or object definition.

An inserted implicit conversion must be in scope as a single identifier or be associated with the source or target type of the conversion.

"Associated with" means found in the companion object of.

Use at least one role-determining name within the type of an implicit parameter.

Bad:

```
(implicit orderer: (T,T) => Boolean)
```

Good:

```
(implicit orderer: T => Ordered[T])
```

Preferred Prompt, not String.

- Integrating a new class into an existing class hierarchy.
- Creating DSLs, syntax-like extensions. You can't add your new "syntax" (e.g. `->`) to `Any`

Return `Option [TupleN]` for some N, since `Tuple` is a `Seq`.

For example, return
`Option[(String,Seq[String])]`