```r
#=============================================================================#
# The trivial pairing scheme:
# Enumerate all possible pairs.
.all_pairs <- function(N){
  purrr::map_dfr(1:N, function(i, N){data.frame(i = rep(i, N), j = 1:N)}, N)
}
#=============================================================================#
# The consecutive pairing scheme:
# Enumerate all possible consecutive/neighboring pairs. Ensures no linear combiantions.
.consecutive_pairs <- function(N){
  purrr::map_dfr(2:N, function(i){data.frame(i = i, j = as.integer(i - 1))})
}
#=============================================================================#
# The lower-triangular pairing scheme:
# Enumerate the pair combinations given N items with i > j.
.unique_pairs_lower <- function(N){
  is <- do.call("c", purrr::map(1:N, function(i){rep(i,N)}))
  js <- rep(1:N, N)
  # Helper function: selects elements only if they are lower triangular
  .isLowerTri <- function(i, j){if(i > j){ c(i = i, j = j) }}
  pairs <- do.call("rbind",purrr::map2(is, js, .f = .isLowerTri))
  data.frame(pairs)
}
#=============================================================================#
# The upper-triangular pairing scheme:
# Enumerate the pair combinations given N items with i < j.
.unique_pairs_upper <- function(N){
  is <- do.call("c", purrr::map(1:N, function(i){rep(i,N)}))
  js <- rep(1:N, N)
  # Helper function: selects elements only if they are lower triangular
  .isUpperTri <- function(i, j){if(i < j){ c(i = i, j = j) }}
  pairs <- do.call("rbind",purrr::map2(is, js, .f = .isUpperTri))
  data.frame(pairs)
}
```