|  |  |  |
| --- | --- | --- |
| **ACTIES OP HET SCHERM** | **VOICE-OVER** | **DUUR** |
|  | Writing modular code facilitates code reuse. If you have to copy and paste procedures when you write code, that is something you will come to regret. If you want to use a procedure in multiple applications, it is much better to define it in a module that can be used in multiple compilation units, even across applications.  Modules also allow you to package your code more conveniently, helping you to keep procedures and even data that belongs together into a single compilation unit. |  |
| 1. Define user defined type 2. Add module 3. Add implicit none 4. Save file | 1. To compute descriptive statistics such as the mean value and standard deviation it suffices to keep track of the sum of the values, the sum of the square of the values and the number of values. We can define this as a user defined type with three elements corresponding to these aggregate values. 2. This user defined type is defined in its own compilation unit, a module, so that it can be reused in multiple applications. 3. It is good practice to declare everything that is defined in this module as private, so it can not be used outside of the module. 4. Everything that we want to use in other compilation units should be declared public explicitly. 5. Note that again, we add “implicit none” to ensure that all variables have to be declared explicitly. |  |
| 1. Add subroutine | 1. We can add the definition of a subroutine that takes the statistics and a value, and updates the aggregate values appropriately. 2. Again, this subroutine has to be declared public. |  |
| 1. Show program unit 2. Use module 3. Declare variable 4. Call subroutine | 1. In the program unit, we use the module, and restrict that to the type definition and the subroutine only. 2. It can than be used to declare the variable “stats”. 3. The subroutine can be used to update the statistics. 4. Finally, the mean value is computed using the aggregated values in the “stats” variable. 5. However, this is a bit messy, as well as the way the “stats” variable is initialized. |  |
| 1. Add initialization subroutine 2. Add mean functions 3. Save file | 1. We can add a subroutine to accomplish this. It should of course be declared a public procedure. 2. Also the way the mean is not so nice, so a function to do that is a lot neater. |  |
| 1. Add call to initialization subroutine 2. Add call to mean function 3. Save file | 1. In the program compilation unit, we can now use the initialization subroutine as well as the function to compute the mean.. |  |
| 1. Add validation subroutine 2. Replace explicit check in mean function by call to validate subroutine. | 1. In practice, we would add a function to compute the standard deviation, which would have to do error handling in case the number of data values is less than 2. 2. We can add a subroutine to check the number of data values for both the function to compute the mean and the one for the standard deviation. 3. This subroutine will only be called from within the module, so it should not be declared public. |  |
|  | As you can see, using modules helps to make code easier to maintain since the code is localized in a single compilation unit. It also promotes code reuse since it is easy to use it from multiple compilation units. |  |
| **TOTALE DUUR** | | *Maak je screencast niet langer dan ca. 6 minuten.* |