|  |  |  |
| --- | --- | --- |
| **ACTIES OP HET SCHERM** | **VOICE-OVER** | **DUUR** |
| Start in empty terminal window | When we were discussing the if-statement, we wrote a function that returns “true” if the year given as an argument is a leap year. Although the function did its job, there are quite some details to add on function declarations. Lets dive in. |  |
| 1. open file in editor 2. write implicit none 3. write variable declaration, assignment and update function call 4. write print statement after call to show value of year 5. assign to year in function 6. save file | 1. Like “program”, a function is a compilation unit. Even if the function definition is contained in the program unit, we still have to declare “implicit none” to make sure that all variables in the function unit have to be declared explicitly. 2. In Fortran, variables are passed by reference. That means that you can actually change the value of a variable that is passed to the function in the calling context. Let’s do a little experiment. We rewrite the program so that the value of a variable is passed to the function “is\_leap\_year”. 3. We also print the value of year after the call to “is\_leap\_year”. 4. In the function, we simply assign a new value to the argument year. As such, that is not a good idea, but this could happen accidentally in a more complex setting. |  |
| 1. switch to terminal 2. compile file 3. run application | 1. When we compile the source and 2. run the application, we see that the variable “year” in the program unit was modified. |  |
| 1. Switch to editor 2. Add intent 3. Save file | 1. Fortran offers some features that help avoid this kind of mistakes. We can declare the intent of the variable. If we only want to use its value, the intent is “in”. If we want to use the value, but also change it, the intent is “inout”. If we only pass a value to the caller and don’t require the value of the argument in the function, the intent is “out”. 2. In this case, we only want to use the value of the variable, so we declare the intent of the argument “year” as “in”. 3. So if the year is divisible by 4, the function should return true, else it should return false. |  |
| 1. Switch to terminal 2. Compile file | 1. Great, now we get a compiler error warning us that the argument “year” is being assigned to while we declared its intent “in”.. |  |
| 1. Switch to editor 2. Remove assignment to year 3. Add optional argument 4. Add present 5. Add call in program 6. Add test 7. Save file | 1. Lets remove the “accidental” assignmentlongs to “year”. 2. It would be nice if we had some better error handling when the argument is a year before the Gregorian calendar was introduced. We will add an optional argument “has\_error”. The type is “logical” and it will be set to true when the year is before 1582. This argument will be set to a value, so its intent is “out”. It is also optional, so we don’t break code that already used the “is\_leap\_year” function. This is declared using the keyword “optional” in the argument declaration. 3. Since the “has\_error” argument is optional, we need to check whether it was provided when the function was called. This can be done using the intrinsic function “present”. If the argument Is present, we assign the appropriate logical value. 4. To test this, lets add another call to “is\_leap\_year” and test the value assigned to “has\_error”. |  |
| 1. Switch to terminal 2. Compile file 3. Run application | 1. Now we can control what to do when something might go wrong when calling the “is\_leap\_year” function. |  |
| 1. Switch to editor 2. Remove logical 3. Add result 4. Add declaration of “is\_leap” 5. Save file | 1. Although the current function signature works, it is a little awkward to assign the return value to the function name. This can be avoided using the “result” keyword to declare a variable that will hold the return value. 2. Of course, we now have to declare the result variable as well. |  |
|  | Of course this didn’t change anything at a fundamental level, however, it feels somewhat more comfortable to some. Examples of the use of “inout” and of subroutine signatures will be shown later. |  |
| **TOTALE DUUR** | | *Maak je screencast niet langer dan ca. 6 minuten.* |