Operator FORMULA

1 Goal

To define a formula in actual value or complex starting from its mathematical expression.

The formula will be usable in a further order like argument of type function/formula or evaluated with particular values of the variables.

In many applications, one can tabuler this formula for particular values by the order CALC_FONC_INTERP [U4.32.01] which produces a concept of the type function or fonction_c like DEFI_FONCTION [U4.31.02] or DEFI_NAPPE [U4.31.03].

2 Syntax

```
F = FORMULA    (

       ♦  NOM_PARA      =  name of the parameters                          [l_K8]
       ♦  /  VALE       =  """ definition of the real formula """      [K]
          /  VALE_C     =  """ definition of the formula complexes """    [K]

        ◊  arg_1 = val_1,
          …
        ◊  arg_ NR = val_ NR

                )
```

F is of type formula or formule_c .

3 Operands

3.1 Definition of the function

The body of the function is an algebraical expression Python represented by a character string. It must be appraisable: i.e. to respect syntax Python and the functions, constants or other objects necessary to its evaluation must defined in argument.

Example:

```
alpha = 1.23
forms = FORMULA (NOM_PARA=' X', VALE=' sin (X) * alpha', alpha=alpha)
```

The expression of the formula is 'sin (X) * alpha'. To evaluate it, X is the variable of the formula. The function sin is provided by the module maths (see below). The constant alpha is defined in the command file. It is thus necessary to define it in argument and to provide the value of it. Thus, the formula could be evaluated where that it is.

Caution

The functions, classes (and other Python objects) defined in the command file are not available in CONTINUATION (Python cannot the "pickler").
So that the formulas using this kind of complementary arguments can be evaluated again in CONTINUATION, it is necessary to redefine the functions, classes (and other Python objects) in CONTINUATION and to declare them with the formulas by setContext.

If one uses VALE, the produced formula is with actual value (concept of the type formula). If one uses VALE_C, the formula is with complex value (concept of the type formule_c).
In both cases, the parameters are real. The names of the parameters necessary to the evaluation of the formula are provided behind the keyword NOM_PARA.

In the event of error of syntax, it is the language Python which transmits the error message and not *Code_Aster* itself.

Notice

The order of the parameters (keyword NOM_PARA) is important. If one creates a formula with two parameters in order to produce a tablecloth, the first parameter is the parameter of the tablecloth, the second is the parameter of the functions composing the tablecloth.

3.2 Functions standards

In addition to the ordinary algebraical signs + -/***, functions standards (buil are also availableTins): min, max, ABS, float…

Attention, the sign of division indicates real division here:
1 / 2 = 0.5
If one wishes to make a whole division operation, should be used the //operator:
1 //2 = 0

3.3 Mathematical functions

All them functions of the module maths of Python by defaults are imported. They are thus directly usable in the body of the formulas.

http://docs.python.org/lib/module-math.html

sin	sinh
cos	cosh
tan	tanh

```
atan        sqrt
atan2       log
asin        log10
acos        exp
```

Moreover, the constant pi, same module, is also available.

Caution:

The goniometrical functions are thus those of Python and expect angles expressed in radians. It is necessary to be vigilant on coherence with the simple keywords ANGL_* process control language which requires angles in degrees in general.

4 Examples of use

For various examples one will refer to the case test ZZZZ100A.

4.1 A formula is used like a tabulée function

Definition of the formula Sia :

```
SIa = FORMULA (NOM_PARA=' X', VALE=' sin (X) ')
```

Equivalent tabulée function IF :

```
LR  = DEFI_LIST_REEL (  BEGINNING = 0. ,
                 INTERVAL = _F (  JUSQU_A = pi, NOT = 0.01)

IF  = CALC_FONC_INTERP (FUNCTION  = SIa,
                        LIST_PARA = LR,
                        NOM_PARA  = 'X',
                        NOM_RESU  = 'DEPL',   )
```

To thus define a function tabulée starting from an interpretable formula, to see CALC_FONC_INTERP [U4.32.01].

Use of IF or of SIa in a simple keyword expecting a function or a formula:

```
champ=CRÉA_CHAMP (… AFFE = _F (… VALE_F  = IF or SIa,   ) )
```

4.2 A formula can be evaluated like a reality

In the body of the command file:

```
SIa = FORMULA (NOM_PARA=' X', VALE=' sin (X) ')

X = SIa (1.57)
print SIa (1.57)
```

Behind a simple keyword expecting a reality:

```
LR = DEFI_LIST_REEL (DEBUT=SIa (0.),
                     INTERVALLE=_F (JUSQU_A=SIa (pi/2.), PAS=0.01))
```

In another formula:

```
SIb = FORMULA (NOM_PARA=' X', VALE=' X*SIa (5.) ', SIa=SIa )
```

4.3 To call upon a formula or a function in another formula

```
SIa = FORMULA (NOM_PARA=' X', VALE=' sin (X) ')
```

Attention to think of putting the argument (X here) in the call to the function SIa :

```
SIb = FORMULA (NOM_PARA=' X', VALE=' X*SIa (X) ', SIa=SIa)
```

4.4 Formula with several parameters

```
NAP = FORMULA (NOM_PARA=('AMOR', 'FREQ'),
               VALE= ''' (1. /((2.*pi*FREQ) ** 2 - OMEGA ** 2) ** 2
                       + (2.*AMOR*2.*pi*FREQ*OMEGA) ** 2) ''',
               OMEGA=OMEGA)
```

In this example, one defines a formula in 2 parameters. Taking into account the length of the expression, she is written for more convenience on several lines with triple quotes to delimit it. The constant pi is constant a standard (cf paragraph [§3.2]), the constant OMEGA must be provided explicitly.

In the actual position, only the formulas of $\mathbb{R}$ in $\mathbb{R}$ or $\mathbb{C}$ are possible: only one produced scalar.

4.5 Formula resulting from programming of function Python

One can refer in a formula to functions programmed in Python, which authorizes formulas much more complex than of algebraic simple expressions.
For example a function of HeavIside:

$$HEAVYSIDE(x)=\begin{cases}0. si\ x<0.\\1. si\ x\geqslant 0.\end{cases}$$

The function Python is programmed as follows:

```
def Heaviside(X):
    yew X < 0.:
        return 0.
    else:
        return 1.

F_HVS = FORMULA (NOM_PARA=' INST',
                 VALE=' Heaviside (INST) ',
                 heaviside=heaviside)
```

Caution:

The use of programming Python in the command file (here method Heaviside) is incompatible with the edition of this file in graphic mode with AsterStudy.

So that the formula F_HVS can be evaluated in CONTINUATION, it is necessary to redefine the function of HeavIside and to declare it with the formula:

```
def Heaviside(X):
    yew X < 0.:
        return 0.
    else:
        return 1.

F_HVS.setContext (dict (heaviside=heaviside) )
```

4.6 Example of definition of formulas in a loop Python

When one defines, in a loop, formulas whose expression depends on the index of the loop, one sees all the interest to pass the constants explicitly.

Example:

```
for I in arranges (3):
    FO [I] = FORMULA (VALE=' cos (i*INST) ', NOM_PARA=' INST', i=i)
    CH [I] = CREA_CHAMP (OPERATION=' AFFE',…, VALE_F=FO [I])
```

With these instructions, one defined 3 formulas which have all the same expression.
With each iteration, a different value is stored for the argument `I`. When the formula is evaluated for then affecting the values of the field, there is no ambiguity on the value of `I`. It is the value attached to the formula during its creation, and not the current value of `I`.
Indeed, in mode `PAR_LOT=' OUI'`, all the orders are create, and then only, carried out. In this mode, `I` is worth 2 during the execution itself of the orders.