

# **PRADIS**

**НАПИСАНИЕ ПЛАГИН-ОБЪЕКТОВ НА ЯЗЫКЕ PYTHON**

**ПРОГРАММНЫЙ КОМПЛЕКС ДЛЯ АВТОМАТИЗАЦИИ  
МОДЕЛИРОВАНИЯ НЕСТАЦИОНАРНЫХ ПРОЦЕССОВ В  
МЕХАНИЧЕСКИХ СИСТЕМАХ И СИСТЕМАХ ИНОЙ  
ФИЗИЧЕСКОЙ ПРИРОДЫ**

**ВЕРСИЯ 4.3**

# 1. Содержание

1. Содержание.....	2
0. Введение.....	3
0. Структура кода.....	4
0. Детали кода.....	5
<a href="#">3.1 Параметры метода Execute.....</a>	<a href="#">5</a>
3.1.1 Модели.....	5
3.1.2 ПРВП.....	5
<a href="#">3.2 Функции библиотеки S000J.....</a>	<a href="#">5</a>
<a href="#">3.3 Вывод ошибок моделей.....</a>	<a href="#">8</a>
Примеры.....	8
Особенности программирования моделей и ПРВП на Питоне.....	10

## **0. Введение.**

В этом документе говорится о правилах написания плагинов объектов (моделей и првп) на языке Питон. О том, как их потом подключать к системному каталогу PRADIS, говорится в документе "Использование утилиты PARM". В нём так же упоминается, что непосредственному коду на питоне должны предшествовать некоторые комментарии, но здесь о них речь не пойдёт.

## 0. Структура кода.

Первая строка должна быть следующего вида:

```
from pradis.ppl.model import *
```

Если Вы пишете првп, то "model" нужно заменить на "ovp".

Далее должен быть объявлен класс. Его название и будет именем Вашего объекта. Этот класс должен наследоваться от класса *model* (если Вы пишете модель) или *ovp* (если Вы пишете првп). Например:

```
class MyOVP (ovp):
```

Далее должен быть переопределён метод *Execute*. Он имеет входные параметры. Для моделей и првп они разные. Определение метода для модели должно выглядеть так:

```
def Execute(COMMON, I, Y, X, V, A, PAR, NEW, OLD, WRK):
```

А для ПРВП так:

```
def Execute(COMMON, XOUT, PAR, WRK, DOF):
```

Далее Вы пишете сам код, который реализует логику работы Вашего объекта. Тут пользователь имеет полную свободу пользоваться всеми средствами языка Питон.

Выход из метода *Execute* должен быть строго определённым. Для моделей он выглядит так:

```
res = return_result(COMMON, I, Y, X, V, A, NEW, OLD, WRK)
return res
```

Для ПРВП так:

```
res = return_result(COMMON, XOUT, WRK)
return res
```

## 0. Детали кода.

Эта секция посвящена объяснению некоторых деталей написания кода, правил и возможностей.

### 3.1 Параметры метода *Execute*.

Здесь мы дадим краткое пояснение параметрам метода *Execute*.

#### 3.1.1 Модели.

Объявление метода для моделей выглядит так:

```
def Execute(COMMON, I, Y, X, V, A, PAR, NEW, OLD, WRK):
```

Здесь параметры имеют следующие значения:

- **COMMON**: структура, содержащая значения непоименованной комон-области фортрана.
- **I**: вектор сил (моментов) для элемента.
- **Y**: якобиан модели элемента.
- **X**: вектор перемещений узлов размерности EXT+ENT. Не используется при ADR=2, ADR=3.
- **V**: вектор скоростей узлов размерности EXT+ENT. Не используется при ADR=3.
- **A**: вектор ускорений узлов размерности EXT+ENT.
- **PAR**: массив параметров модели.
- **NEW**: вектор "нового состояния" модели.
- **OLD**: вектор "старого состояния" модели.
- **WRK**: рабочий массив для модели элемента.

#### 3.1.2 ПРВП.

Объявление метода для ПРВП выглядит так:

```
def Execute(COMMON, XOUT, PAR, WRK, DOF):
```

Здесь параметры имеют следующие значения:

- **COMMON**: структура, содержащая значения непоименованной комон-области фортрана.
- **XOUT**: рассчитываемая выходная переменная или вектор рассчитываемых выходных переменных.
- **PAR**: массив параметров ПРВП.
- **WRK**: рабочий массив для ПРВП.
- **DOF**: массив со значениями степеней свобод.

### 3.2 Функции библиотеки **S000J**.

При написании плагинов объектов на питоне пользователю доступны функции библиотеки S000J, которые часто используются, например, в моделях на фортране. Все функции, которые были доступны на фортране, теперь доступны из питона. В этом разделе мы подробно расскажем, как ими пользоваться.

Прежде всего надо подключить необходимую библиотеку к Вашему питон-файлу:

```
from s000j import *
```

После этого создать объект соответствующего класса:

```
sj = S000J()
```

Теперь Вам будут доступны методы этого класса, которые называются как раз по названиям нужных функций (S0001, S0002, ...). И тут есть один важный момент, на который стоит обратить внимание. Он связан с передачей параметров в методы и получением результатов из них.

В фортране в функции передавался набор переменных, одни из которых служили передачи данных в функцию, а другие из неё. Вызов этих функций из питона отличается тем, что выходные параметры и массивы в них нужно передавать в виде списков, предварительно занесенных в массивы специально разработанными для этого функциями. Приведём пример.

Вот одна из таких функций на фортране:

```
C      Программа выдает значение функции и ее производной
C      при заданном значении аргумента.
C
C      Дата создания программы          03/10/95 08:42am
C      Дата последней корректировки     09/14/95 11:51am
C
C      X      - заданное значение аргумента
C      F      - значение функции при заданном значении аргумента
C      DFDX   - значение производной функции при заданном
C              значении аргумента
C      TABL   - таблица значений функции (попарно - значение
C              аргумента/ значение функции)
C      N      - длина таблицы.
C      NPNT   - массив номеров точек, между которыми
C              находится текущая точка
C      Если значение аргумента меньше минимально определенного
C      или больше максимально определенного в таблице, то
C      значение функции вычисляется экстраполяцией.
C
C      SUBROUTINE S0008 (F, DFDX, X, TABL, N, NPNT )
C
C      REAL      * 8  X,          F,          DFDX,          TABL (1)
C      INTEGER   * 4  N,          NPOINT,  J,          NPNT (1)
```

Ее вызов из модели на Питоне будет выглядеть следующим образом:

```
F_a = CreateDoubleArrayFromList([0.])
DFDX_a = CreateDoubleArrayFromList([0.])
l = []
for i in range(REZPAR+1, len(PAR)):
    l.append(PAR[i])
TABL_a = CreateDoubleArrayFromList(l)
NPNT_a = CreateIntArrayFromList([0,0])
sj.S0008(F_a, DFDX_a, DX, TABL_a, NPOINT, NPNT_a)
l = CreateListFromDoubleArray(F_a, 1)
F = l[0]
```

```

l = CreateListFromDoubleArray(DFDX_a, 1)
DFDX = l[0]
NPNT = CreateListFromArray(NPNT_a, 2)
DeleteArray(F_a)
DeleteArray(DFDX_a)
DeleteArray(TABL_a)
DeleteArray(NPNT_a)

```

Сдесь F\_a и DFDX\_a будут массивами, содержащими по одной выходной переменной. TABL\_a будет содержать входной массив, сформированный из списка PAR. NPNT\_a будет выходной массив состоящий из двух целых переменных. DX и NPOINT являются простыми входными параметрами и следовательно не нуждаются в каких либо преобразованиях. Далее идет вызов функции S0008:

```

sj.S0008(F_a, DFDX_a, DX, TABL_a, NPOINT, NPNT_a)

```

После этого, выходные параметры и массивы надо снова преобразовать в питоновский формат.

Например из массива F\_a мы сначала создаем список используя функцию CreateListFromDoubleArray:

```

l = CreateListFromDoubleArray(F_a, 1)

```

А затем берем первый элемент этого списка, что бы получить значение рассчитанной переменной F:

```

F = l[0]

```

Так же точно мы поступаем с переменной DXDF:

```

l = CreateListFromDoubleArray(DFDX_a, 1)
DFDX = l[0]

```

А массив целочисленных выходных значений NPNT\_a преобразовываем в список NPNT:

```

NPNT = CreateListFromArray(NPNT_a, 2)

```

Следует обратить внимание, что для преобразований реальных и целых массивов используются различные функции. Для реальных используются функции:

```

CreateDoubleArrayFromList
CreateListFromDoubleArray

```

А для целых используются функции:

```

CreateIntArrayFromList
CreateListFromArray

```

В конце всего разработчик модели обязательно должен освободить память, отведенную под массивы функцией DeleteArray:

```

DeleteArray(F_a)
DeleteArray(DFDX_a)
DeleteArray(TABL_a)
DeleteArray(NPNT_a)

```

### 3.3 Вывод ошибок моделей.

При написании плагин моделей на питоне пользователь может использовать доступный набор шаблонов для сообщений об ошибках. Полный список шаблонов можно увидеть в документе "Доступные ошибки моделей". Здесь мы расскажем, как пользоваться доступными шаблонами.

Допустим, Вы захотели вывести ошибку, используя шаблон 1003:

```
1003 08 E (M 004) (/T4, 'Некорректное значение
      параметра: ',/, T8, 'параметр ', F6.0, ' должен быть >=
      ', G11.5,/, T8, 'параметр ', F6.0, ' = ', G11.5)
```

Как видите, этот шаблон требует 4 числа: 2 целых и 2 действительных (встречающиеся F6.0 - это целые числа, G11.5 - действительные). Если задать эти числа значениями 1, 2.2, 1, 1.1, то на экране это будет выглядеть так:

```
Некорректное значение параметра:
параметр      1. должен быть >=   2.2000
параметр      1. = 1.1000
```

Чтобы использовать шаблон, в питон-файле Вашей модели нужно сделать следующее. Подключить необходимые библиотеки:

```
from PradisLog import *
from array import *
```

Библиотека *PradisLog* поставляется с комплексом PRADIS, а *array* является стандартной питоновской библиотекой. Далее нужно создать объект нужного класса:

```
pl = PradisLog()
```

Теперь в том месте, где Вам нужно вывести ошибку, надо написать следующее:

```
a = array ('d', [1, 2.2, 1, 1.1])
pl.perr (1003, 4, a.buffer_info()[0])
```

При выполнении этих строк на экране появится именно то, что указывалось выше. Первая строка создаёт массив действительных чисел, который заполняется теми числами, которые Вы хотите видеть в шаблоне при выводе сообщения. Количество этих чисел должно строго совпадать с количеством требуемых в шаблоне. Вторая строка использует метод *perr* класса *PradisLog*. В его параметрах следует указывать номер шаблона, количество чисел в шаблоне и адрес начала массива параметров.

В примере мы назвали переменные "a" и "pl", но пользователь, конечно, может называть их по-своему.

### Примеры.

Здесь приведём пример питон модели. Это уже имеющаяся в библиотеке комплекса модель MD. Вот как она могла бы выглядеть на питоне:

```
# Библиотека нужная для написания модели
```



```

from pradis.ppl.model import *

# Библиотеки для вывода ошибок
from PradisLog import *
from array import *

# Библиотека для использования функций S000J
from s000j import *

# Объявляем класс MD и указываем, что это модель
class MD (model):

    # Переопределяем метод Execute (обязательно)
    def Execute(COMMON, I, Y, X, V, A, PAR, NEW, OLD, WRK):

        # Далее пишем суть модели
        # Первый проход
        if COMMON.NEWINT == 1:

            # Создаём объект для использования функций S000J
            sj = S000J()

            # Используем нужную функцию
            d = sj.S0005 (0.1, 0.2, 0.3, 0.4, 0.5, 1)
            print d

            # Создаём объект для вывода ошибок
            pl = PradisLog()
            ERR = 0

            if PAR[1] < 0.:
                ERR = 1
                if COMMON.SYSPRN > 0.:
                    # Выводим ошибку по шаблону 1003
                    a = array ('d', [1, 0, 1, PAR[1]])
                    pl.perr (1003, 4, a.buffer_info()[0])

            if PAR[2] < 0.:
                ERR = 1
                if COMMON.SYSPRN > 0.:
                    # Выводим ошибку по шаблону 1003
                    a = array ('d', [2, 0, 2, PAR[2]])
                    pl.perr (1003, 4, a.buffer_info()[0])

            if ERR == 1:
                if COMMON.CODE < 100.:
                    COMMON.CODE = 100.

            # Выходим из модели
            res = return_result(COMMON, I, Y, X, V, A, NEW, OLD,
WRK)

            return res

I[1] = A[1] * PAR[1]
I[2] = A[2] * PAR[1]
I[3] = A[3] * PAR[2]

Y[1] = PAR[1]
Y[2] = 0.
Y[3] = 0.
Y[4] = 0.
Y[5] = PAR[1]
Y[6] = 0.

```

```

Y[7] = 0.
Y[8] = 0.
Y[9] = PAR[2]

# Выходим из модели
res = return_result(COMMON, I, Y, X, V, A, NEW, OLD, WRK)
return res

```

## Особенности программирования моделей и ПРВП на Питоне.

Необходимо обратить внимание на то, что тип переменных в Питоне определяется при их инициализации. Поэтому если вы хотите работать с реальной переменной, а присвоите ей целое значение, например:

```
R = 2
```

То R у вас будет целой, а не реальной переменной. Для правильной работы, задавая реальную переменную ее надо обязательно инициализировать реальной константой, т.е. константой, содержащей в себе десятичную точку, или другой реальной переменной. Например:

```
R = 2.
```

Эти моменты очень важно учитывать при формировании списков выходных значений. Так как в моделях все входные параметры передаются в виде списков реальных чисел, то и возвращаться из моделей должны списки реальных чисел. Поскольку Питон позволяет в один и тот же список заносить элементы разных типов, то разработчик по ошибке может занести в выходной список целочисленную константу или переменную, что неминуемо приведет к ошибке.