



Функції в Scilab

Функції в Scilab відіграють роль підпрограм. Виділення послідовності інструкцій в окрему функцію, яку можна повторно використовувати, є однією з найбільш поширених задач при роботі зі Scilab. Найпростіший синтаксис виклику функції виглядає наступним чином:

outvar = myfunction (invar)

Значення кожного з трьох елементів виклику функції наведено нижче:

myfunction - представляє собою найменування функції, що викликається;

invar - позначає вхідні аргументи;

outvar - відповідає вихідним аргументам.

Значення змінних, зазначених при виклику в якості фактичних параметрів, функція змінити не може.

Кількість вхідних і вихідних аргументів функції необмежена. Синтаксис виклику функції з фіксованим числом аргументів такий:

[o1 , ... , on] = myfunction (i1 , ... , in)

Список вхідних аргументів обмежується круглими дужками, а вихідних - квадратними. Назви змінних в списках відокремлюються один від одного комами ",".

Ключові слова та інструкції Scilab, що використовуються при роботі з функціями.

function – відкриває визначення функції;

endfunction – завершує визначення
функції;

argn – кількість вхідних і вихідних
аргументів в даному виклику функції;

varargin – вектор, що представляє змінне
число вхідних аргументів функції


varargout – вектор, що представляє змінне число вихідних аргументів функції;

fun2string – генерує текстове визначення (вихідний код) функції;

get_function_path – повертає шлях до файлу вихідного коду функції;


getd – відображає повний список функцій, визначення яких зберігаються в заданому каталозі файлової системи;

head_comments – відображає коментарі до функції;



listfunctions – відображає властивості функцій, які викликалися раніше в даній сесії;

macrovar – повертає списки вхідних і вихідних параметрів функції, а також використаних в тілі функції зовнішніх змінних, викликів інших функцій і локальних змінних.



Функція, як правило, призначена для неодноразового використання, вона має вхідні параметри і не виконується без їх попереднього задання. Розглянемо декілька способів створення функцій в Scilab.

Перший спосіб це застосування оператора `deff`, який в загальному вигляді можна записати так:

deff ('[ім'я1, ..., ім'яN] = ім'я_функції
(змінна_1, ..., змінна_M)', 'ім'я1 = вираз1; ...;
ім'яN = виразN')

де ім'я1, ..., ім'яN - список вихідних параметрів, тобто змінних, яким буде присвоєно кінцевий результат обчислень (параметрів може бути від 1 до N), ім'я_функції - ім'я з яким ця функція буде викликатися, змінна_1, ..., змінна_M - вхідні параметри (параметрів може бути від 1 до M).

Далі наведено найпростіший спосіб застосування оператора `deff`. У наступному прикладі створена функція **`myfunction`**, яка приймає параметр **`x`**, множить його значення на **2** і повертає результат в якості вихідного параметра **`y`**:

```
-->deff('y=myfunction(x)', 'y=2*x');
```

```
-->x=3;y=myfunction(x)
```


```
y =
```

```
6.
```

Другий спосіб створення функції це застосування конструкції виду:

```
function [ім'я1,..., ім'яN] = ім'я_функції (змінна_1,  
..., змінна_M)  
<тіло функції>  
endfunction
```

де ім'я1, ..., ім'яN - список вихідних параметрів, тобто змінних, яким буде присвоєно кінцевий результат обчислень (параметрів може бути від 1 до N), ім'я_функції - ім'я з яким ця функція буде викликатися, змінна_1, ..., змінна_M - вхідні параметри (параметрів може бути від 1 до M).



Всі імена змінних всередині функції, а також імена зі списку вхідних і вихідних параметрів сприймаються системою як локальні, тобто ці змінні вважаються визначеними тільки всередині функції.

Взагалі кажучи, функції в Scilab відіграють роль підпрограм. Тому доцільно набирати їх тексти в редакторі і зберігати у вигляді окремих файлів. Причому ім'я файлу повинно обов'язково збігатися з ім'ям функції. Розширення файлам-функціям зазвичай присвоюють sci або sce.

У наступному прикладі створена функція **myfunction**, яка приймає параметр **x**, множить його значення на **2** і повертає результат в якості вихідного параметра **y**:

```
function y = myfunction ( x )
```

```
y = 2 * x
```

```
endfunction
```

Рядок **function y = myfunction (x)** є заголовком функції, в той час як тіло функції містить єдину інструкцію **y=2*X**. У загальному випадку тіло функції може включати довільне число команд. У Scilab існує, щонайменше, три способи визначити таку функцію:

1) По-перше, можна ввести тіло функції безпосередньо в консолі Scilab, інструкція за інструкцією. Зустрівши запис на зразок `function y = myfunction(x)`, інтерпретатор переходить в режим очікування тіла функції. Завершується введення командою `endfunction`, після чого Scilab повертається в звичайний режим.

2) Більш зручним варіантом є визначення функції в окремому файлі. Цей спосіб застосовується в більшості випадків.

3) Також для завантаження функції можна використовувати команду **exec**.

Припустимо, що визначення функції розміщується у файлі:

C:\myscripts \examples-functions.sce.

Для завантаження застосовується команда **exec**, як показано в наступному фрагменті:


```
-->exec("C:\myscripts\examples-functions.sce ")  
--> function y = myfunction ( x )  
--> y = 2 * x  
--> endfunction
```

Функція **exec** призначена для виконання інструкцій, які містяться в деякому файлі, так, якщо б вони вводилися безпосередньо в консолі **Scilab**. При цьому в консолі відображається кожен рядок алгоритму. Якщо файл містить велику кількість команд, відображення кожної з них може виявитися небажаним.

Щоб цього уникнути, після інструкції `exec` необхідно поставити ";" :

```
exec("C:\myscripts\examples-functions.sce");
```

Того ж результату можна досягти, вибравши пункт **Виконати** у меню **Файл** в **Scilab** (**Execute file into Scilab**). Після того як функція створена, її можна використовувати подібно будь-якій іншій команді Scilab.

Зауважимо, що присвоєння вихідному аргументу y значення (в даному випадку $y=2*x$) є обов'язковим. Для того щоб переконатися в цьому, розглянемо наступний приклад, де значення присвоюється змінній z , а не вихідному параметру y :

```
function y = myfunction (x)
```

```
z = 2 * x
```

```
endfunction
```

Спробуємо тепер викликати цю функцію, задавши їй значення 1:

```
--> myfunction ( 1 )
```


```
!-- error 4
```

```
Undefined variable : y
```

```
at line 4 of function myfunction called by :
```

```
myfunction ( 1 )
```


Інтерпретатор **Scilab** повідомляє нам про помилку, оскільки змінна **y** не була ініціалізована в тілі функції.



При вирішенні деякої задачі часто виникає потреба у визначенні більш ніж однієї функції. Часто на практиці для вирішення завдання може знадобитися і декілька десятків функцій. В цьому випадку буде правильно об'єднати функції в бібліотеку.

Бібліотеки функцій

Бібліотека є набором функцій, написаних мовою **Scilab**, які зберігаються в окремих файлах. Якщо набір функцій невеликий і не містить файлів довідки або вихідних текстів на компільованих мовах (таких як C / C ++ або Fortran), об'єднування їх у бібліотеку є досить вдалим варіантом, в іншому випадку слід задуматися про створення модуля.



Розробка власного модуля не представляє труднощів, проте вимагає більш детального знайомства з внутрішньою будовою пакету Scilab. Крім того, модулі також мають у своїй основі бібліотеки, тому для створення перших потрібно розуміння роботи останніх. У багатьох практичних ситуаціях створення бібліотеки є достатнім для організації набору функцій.

Розглянемо створення простої бібліотеки функцій **Scilab**, а також способи її автоматичного завантаження при запуску пакета. Припустимо, що є декілька файлів .sci, що містять визначення функцій мовою **Scilab**. Послідовність дій у цьому випадку така:

- 1) Створити бінарні версії функцій, використовуючи команду **genlib**. Функція **genlib** крім іншого генерує індексні файли;
- 2) Завантажити бібліотеку в **Scilab**, для чого використовується функція **lib**.

Перед тим як приступити до розгляду прикладу, необхідно визначити основні правила створення бібліотек функцій в **Scilab**:

1) Файли, що містять визначення функцій, повинні мати розширення **.sci**. Строго кажучи, ця вимога не є обов'язковою, але допомагає при пошуку скриптів **Scilab** на жорсткому диску комп'ютера.

2) В одному файлі **.sci** можуть бути визначені декілька функцій **Scilab**, однак тільки перша з них буде доступна ззовні. Іншими словами, тільки перша функція, визначена у файлі, вважається загальнодоступною.

3) Файл **.sci** має збігатися з ім'ям загальнодоступної функції в цьому файлі. Наприклад, якщо ім'я функції **myfun**, то файл, що містить її, повинен мати назву **myfun.sci**. Ця вимога є обов'язковою, в іншому випадку функція **genlib** не працюватиме коректно.

Інструкції Scilab, які використовують при роботі з бібліотеками функцій

genlib - створює бібліотеку з функцій, визначення яких розташовані в заданому каталозі;

lib - завантажує бібліотеку.

Тепер перейдемо до прикладу створення конкретної бібліотеки. Припустимо, що ми працюємо на комп'ютері під керуванням ОС Windows. Нехай в каталозі **samplelib** розміщуються два файли:

C:\samplelib\function1.sci:

function y = function1 (x)

y = 1 * function1_support (x)

endfunction

function y = function1_support (x)

y = 3 * x

endfunction

```
C:\samplelib\function2.sci:  
function y = function2 ( x )  
y = 2 * x  
endfunction
```

Для отримання бінарних версій цих функцій використовуємо інструкцію **genlib**.

--> ***genlib (" mylibrary ", "C:\samplelib ")***

Перший аргумент **genlib** представляє назву майбутньої бібліотеки, а другий вказує каталог, де розміщені файли функцій.

Зауважимо, що в даному випадку тільки функції **function1** і **function2** є загальнодоступними, а функція **function1_support** може використовуватися тільки усередині бібліотеки, але не поза нею.

Функція **genlib** генерує і поміщає в каталог **C:\samplelib** наступні файли:

1) **function1.bin**: бінарна версія файлу **function1.sci**,

2) **function2.bin**: бінарна версія файлу **function2.sci**,

3) **lib**: бінарна версія бібліотеки,

4) **names**: текстовий файл, що містить імена всіх функцій в бібліотеці.

Отримані файли ***.bin** і файл **lib** можуть без змін використовуватися версіями **Scilab** для **Windows, Linux і Mac OS**. Відразу ж після виклику **genlib**, дві нові функції стають доступні середовищу **Scilab** і викликаються таким чином:

```
--> Function1 (3)
```

```
ans =
```

```
9.
```

```
--> Function2 (3)
```

```
ans =
```

```
6.
```


Звичайно, кожен раз генерувати бібліотеку заново не потрібно. Готову бібліотеку можна завантажити за допомогою команди **lib**, єдиний аргумент який вказує місце розташування завантаження бібліотеки в файловій системі. Наступний фрагмент ілюструє завантаження створеної нами бібліотеки:

```
--> mylibrary = lib ("C:\samplelib \")
```

```
mylibrary =
```

```
Functions files location : C:\samplelib \.
```

```
function2    function1
```





При великому числі бібліотек, що завантажуються зручно помістити виклики **lib** в стартовий скрипт **Scilab**. У цьому випадку всі зазначені бібліотеки будуть доступні відразу ж після запуску **Scilab**. Файл стартового скрипта розміщується в основному каталозі **Scilab**, шлях до якого можна дізнатися, перевіривши значення змінної **SCIHOME**:

--> **SCIHOME**

**SCIHOME =C:\ Users \ username \
AppData \ Roaming \ Scilab \ scilab
-5.5.0**

Стартовий скрипт **SCIHOME** має ім'я **.scilab** і є звичайним скриптом Scilab, в тому числі може містити коментарі. Для завантаження створеної раніше бібліотеки додамо в файл **.scilab** наступні рядки:



```
// Load my favorite library.  
mylibrary = lib ("C:/samplelib/")
```

В результаті бібліотека **mylibrary** буде завантажуватися щоразу при запуску Scilab.

Керування вихідними змінними

Кожна функція **Scilab** може мати один або декілька вхідних і вихідних аргументів. У найпростішому випадку число вхідних і вихідних аргументів фіксовано, тому використовувати таку функцію неважко. Однак, як ми побачимо далі, навіть найпростіші випадки допускають варіації.

Припустимо, що функція **simplef** визначена з двома вхідними і вихідними аргументами:

```
function [y1 , y2] = simplef ( x1 , x2 )
```

```
y1 = 2 * x1
```

```
y2 = 3 * x2
```

```
Endfunction
```

При виконанні функції можна вказати два, один або жодного вихідного аргументу. Якщо не вказано жодного вихідного аргументу, значення яке повинно було бути присвоєно першому з них, поміщається в змінну **ans**. Можна вказати один вихідний аргумент, який отримає значення **y1**.

Також можна надати функції два аргументи, як це передбачено її визначенням. У наступному прикладі розглядаються всі три варіанти виклику функції **simplef** з різним числом вихідних змінних:

--> simplef (1 , 2)

ans =

2.

-->y1 = simplef (1 , 2)

y1 =

2.

-->[y1 ,y2] = simplef (1 , 2)

y2 =

6.

y1 =

2.

Таким чином, навіть найпростіше визначення функції допускає деяку свободу щодо числа вихідних аргументів. Гнучкішим способом завдання змінного числа вхідних і вихідних аргументів є використання ключових слів **argn**, **varargin** і **varargout**. Ці можливості ми не будемо сьогодні розглядати, але для побудови дійсно гнучких функцій слід мати їх на увазі.

Рівні стеку викликів

Як і в інших мовах програмування, в Scilab виклики функцій можуть бути вкладеними, тобто функція **f** може викликати функцію **g**, а та, в свою чергу, звертатися до функції **h** і т.д. Команди, введені користувачем в консолі **Scilab**, відповідають нульовому рівню стека викликів. Інструкції, які знаходяться в тілі запусненої з консолі функції, становлять перший рівень.

Команди Scilab для роботи зі стеком викликів.

whereami - відображає поточне дерево викликів;

where - представляє поточне дерево викликів у вигляді матриці.

Для ілюстрації ми визначимо три функції **fmain**, **flevel1** і **flevel2**, які викликають одна одну, і використовуємо в **flevel2** інструкцію **whereami**, яка відображає поточний стан стеку викликів:

```
function y = fmain ( x )  
y = 2 * flevel1 ( x )  
endfunction  
function y = flevel1 ( x )  
y = 2 * flevel2 ( x )  
endfunction  
function y = flevel2 ( x )  
y = 2 * x  
whereami ()  
endfunction
```

При виконанні функції `fmain` в консоль буде виведено:

--> `fmain (1)`

whereami called at line 3 of macro `flevel2`

`flevel2` called at line 2 of macro `flevel1`

`flevel1` called at line 2 of macro `fmain`

`ans =`

8.

Як бачимо, відображається три рівні стека викликів, які відповідають викликаним функціям.

Значення рівнів стека викликів в наведеному прикладі такі:

- 1) рівень 0 - глобальний (команди, що вводяться в консолі **Scilab**),
- 2) рівень -1 - в тілі функції **fmain**,
- 3) рівень -2 - в тілі функції **flevel1**,
- 4) рівень -3 - в тілі функції **flevel2**.

Налагоджувальні інструкції Scilab.

Pause – призупинити виконання функції і чекати команд користувача;

Resume – продовжити виконання функції;

Abort – завершити виконання функції.

Інструкція `return`

Інструкція `return` дозволяє завершити виконання функції і повернути управління коду, який викликається. Представлена нижче функція обчислює суму цілих чисел від `istart` до `iend`. При коректних значеннях параметрів для цього викликається функція `sum`, однак якщо значення змінної `istart` від'ємне або якщо умова `istart <= iend` не виконується, значення змінної `y` встановлюється в `0` і функція завершує виконання.

```
function y = mysum ( istart , iend )  
if ( istart < 0 ) then  
y = 0  
return  
end  
if ( iend < istart ) then  
y = 0  
return  
end  
y = sum ( istart : iend )  
endfunction
```


Наступний фрагмент дозволяє переконатися, що функція **mysum** працює саме так, як задумано:

```
--> mysum ( 1 , 5 )
```

```
ans =
```

```
15.
```

```
--> mysum ( -1 , 5 )
```

```
ans =
```

```
0.
```

```
--> mysum ( 2 , 1 )
```

```
ans =
```

```
0.
```

Присутність в тілі однієї функції численних інструкцій `return` вважається поганим стилем програмування, оскільки серйозно ускладнює аналіз вихідного коду такої функції. У цьому зв'язку рекомендується обмежитися використанням `return` лише в тих ситуаціях, де в іншому випадку необхідне було б написання великого обсягу додаткового коду. Як правило ж, функція повинна повертати управління при досягненні останнього рядка.