Язык программирования Питон (Python)

Модули

Модули – файлы, содержащие ***наборы различных функций*** (а также переменных, классов) ***для их повторного использования*** в других программах.

Для использования функций из модуля, его нужно ***импортировать*** командой `import`

(*инициализация происходит только при первом импорте модуля. При этом содержащийся в модуле код исполняется.*)

Доступ к объектам в модуле предоставляется при помощи точки, т.е. `math.sqrt()`.

Пример:

```
import math          # загрузка стандартного модуля математических функций
y=math.sqrt(9)       # обращение к функции из модуля
print(y)              -> 3.0
```

Пример:

```
import numpy as np       # задание псевдонима для модуля
```

Пример:

```
import sys
print('Аргументы командной строки: ', sys.argv, '\n')
print('Переменная PYTHONPATH содержит: ', sys.path, '\n')

import os
print('Текущий каталог программы: ', os.getcwd())
```

Замечания.

Переменная **sys.argv** является списком строк, содержащим ***список аргументов командной строки***.

Переменная **sys.path** содержит ***список имён каталогов, откуда импортируются модули***.

Первая строка в **sys.path** пуста – текущая директория также является частью **sys.path** (*модули, расположенные в текущем каталоге, можно импортировать напрямую. Иначе модули необходимо поместить в один из каталогов, перечисленных в* **sys.path**).

Текущий каталог – это каталог, в котором была запущена программа.

Перечень и описание функций, содержащихся в модуле можно узнать из справки:

Пример:

```
help(math)
help(math.sqrt)
```

Можно ***импортировать отдельную функцию или переменную из модуля***

(*В этом случае обращение к функции или переменной производится без указания имени модуля*)

Пример:

```
from math import sqrt
y = sqrt(9)
print(y)

from math import pi
print('pi=', pi)
```

Замечание. *При импортировании отдельных функций из модулей* ***возможен конфликт их имён*** *с именами функций опреднлённых в программе.*

Для ***импортирования всех объектов из модуля*** используется оператор `from ... import *`.

(*за исключением имён, начинающихся с двойного подчёркивания*)

Замечание. *Следует избегать использования этого оператора, чтобы предотвратить* ***конфликты имён***

Пример:

```
from math import *

print('pi=', pi, '  e=', e)
print('sin(pi/2)=', sin(pi/2), '  cos(pi/2)=', cos(pi/2))
```

Можно заставить ***модуль вести себя по-разному*** в зависимости от того, ***запущен ли он*** сам по себе ***или импортируется*** в другую программу.

В каждом модуле Python определена специальная переменная – **`__name__`**.
Если мы запускаем модуль, то содержимое переменной **`__name__`** будет равно строке **`'__main__'`**,
а *в случае импортирования* – переменная **`__name__`** будет содержать имя модуля.

Пример:

Сохраним следующий код как **`using_name.py`**

```
if __name__ == '__main__':
    print('Эта программа запущена сама по себе.')
else:
    print('Меня импортировали в другой модуль.')
```

– *Если мы запустим данную программу, получим сообщение:*

```
-> Эта программа запущена сама по себе.
```

– *Если в теле другой программы исполнится команда* **`import using_name`**, *получим сообщение:*

```
-> Меня импортировали в другой модуль.
```

Пример: – *Создадим модуль с именем* **prog.py** *и сохраним его в текущей директории:*

```
def func(x):
    return x**2+7

if __name__ == "__main__":
    x=int(input("Введите значение: "))
    print(func(x))
```

– *Если запустить данный файл на исполнение, то он выполнится целиком, в том числе сработает оператор* **if**

– *При импортировании модуля условие не выполняется и* Python *только загрузит в память функцию* **func()**, *которую впоследствии можно вызвать с указанием имени модуля:*

```
import prog

prog.func(5)    -> 32
```

Создание собственных модулей

– создать файл с расширением **.py**, содержащий переменные, функции, классы, написанные на Python;

Замечание. *Модуль можно написать на другом языке программирования (например на языке С), который после компиляции может использоваться стандартным интерпретатором Python.*

– разместить данный файл в текущей директории или в директориях указанных в **sys.path.**

– при импортировании модуля командой **import** указывается его имя без расширения;

– для вызова переменной, функции или класса из модуля их имя необходимо указывать через точку после имени модуля.

Встроенные функции Python на самом деле ***находятся в модуле*** **`__builtins__`**, который загружается в память в начале работы.

Содержание данного модуля можно узнать из справки:

```
help (__builtins__)
```

Функция **`dir()`** – возвращает ***список идентификаторов*** (функции, классы, переменные), определенных в модуле, переданном в качестве аргумента.

(*Если никакого аргумента не передавать, вернёт список имён, определённых в текущем модуле*)

Пример: *(набирать команды в строке консоли)*

```
dir(__builtins__)   # список идентификаторов модуля __builtins__

dir('print')        # получим атрибуты функции  print
dir('str')          # получим атрибуты класса  str

import sys
dir(sys)            # получим список атрибутов модуля  'sys'

import math
dir(math)           # получим список атрибутов модуля  'math'
```

`del a` – удаление имени **`a`** из памяти

Дополнительные замечания:

Модули по своему происхождению делятся на

- ***обычные*** (написанные на Python) и
- модули ***расширения***, написанные на другом языке программирования (как правило, на C).

С точки зрения пользователя они могут отличаться разве что быстродействием или гибкостью. Бывает, что в стандартной библиотеке есть два варианта модуля: на Python и на C.

Набор модулей, посвященных одной проблеме, можно поместить в ***пакет***.

В последних версиях Python модули можно помещать и в **zip-*архивы*** для более компактного хранения.

Загруженный модуль можно ***загрузить еще раз*** (например, если модуль изменился на диске) с помощью функции `reload()`:

Пример:

```
import mymodule
. . .
reload(mymodule)
```

Однако в этом случае все объекты, являющиеся экземплярами классов из старого варианта модуля, не изменят своего поведения.