Модули в Python

По мере того как программы возрастают в размерах, может возникнуть необходимость разбить их на несколько файлов для удобства сопровождения. Для этого в языке Python предусмотрена возможность поместить в файл определения и использовать их в качестве модуля, который можно импортировать в другие программы и сценарии.

Для создания модуля нужно поместить соответствующие операторы и определения в файл, имеющий такое же имя, как у модуля. Файл должен иметь расширение .py.

Библиотеки или модули нужны для того, чтобы расширить возможности Python и упростить написание программ.

Модуль *math*, например, помогает при работе с числами, а модуль *datetime* нужен для работы с датой и временем.

Попробуем импортировать, то есть сделать доступным для использования, математический модуль в нашу программу. Сам импорт осуществляется с помощью команды *import* и названия импортируемого модуля.

```
import math
```

Теперь можно полноценно пользоваться всеми функциями этого модуля. Для этого необходимо написать название этого модуля, точку и название нужной функции из модуля.

Пример подключения модуля:

Например, выводим на экран значение числа π:

```
import math
print (math.pi)
```

или косинус единицы

```
import math
print (math.cos(1))
```

Пример:

```
# файл :   div.py
def divide(a,b):
    q = a/b #Если a и b – целые, то q – целое число
    r = a – q*b
    return (q,r)
```

Для того чтобы использовать этот модуль в других программах, можно воспользоваться оператором import:

```
import div
a, b = div.divide(2305, 29)
```

Оператор import создает новое пространство имен, которое содержит все объекты, определенные в модуле. Для доступа к этому пространству имен можно просто использовать имя модуля в качестве префикса, как в случае div.divide() из предыдущего примера. Для выполнения импорта конкретных определений в текущее пространство имен служит оператор from:

```
from div import divide
a,b = divide(2305,29) # Префикс div больше не нужен
```

Для загрузки всего содержимого модуля в текущее пространство имен можно также использовать такую конструкцию:

```
from div import *
```

Функция dir() выводит на экран содержимое модуля и является полезным средством экспериментирования в интерактивном режиме:

```
>>> import string
>>> dir(string)
['__builtins__', '__doc__', '__file__', '__name__', '_idmap', '_idmapL',
'_lower', '_swapcase', '_upper', 'atof, 'atof_error', 'atoi', 'atoi_error', 'atol',
'atol_error', 'capitalize', 'capwords', 'center', 'count', 'digits', 'expandtabs',
'find',
```

Подключение модуля из стандартной библиотеки

Подключить модуль можно с помощью инструкции import. К примеру, подключим модуль os для получения текущей директории:

```
>>> import os
>>> os.getcwd()
'C:\\Python33'
```

После ключевого слова **import** указывается название модуля. Одной инструкцией можно подключить несколько модулей, хотя этого не рекомендуется делать, так как это снижает читаемость кода. Импортируем модули time и random.

```
>>> import time, random
>>> time.time()
1376047104.056417
>>> random.random()
0.9874550833306869
```

Подключение модуля из стандартной библиотеки

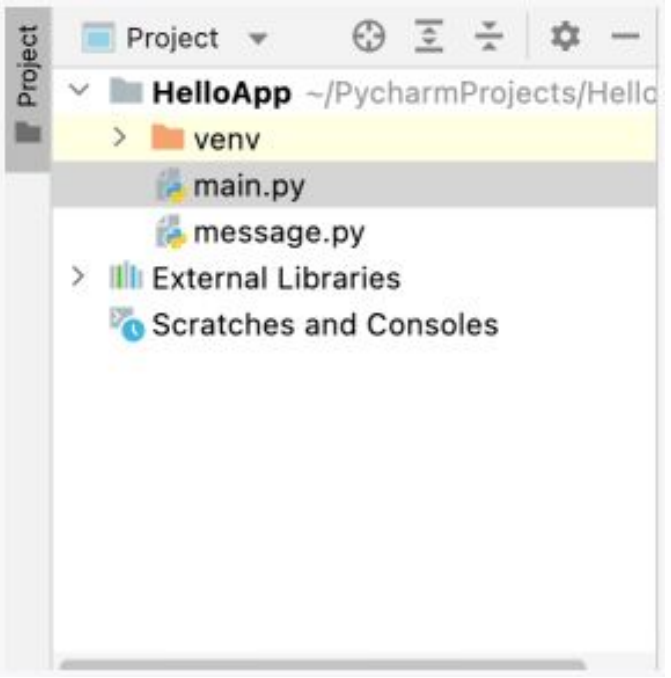
После импортирования модуля его название становится переменной, через которую можно получить доступ к атрибутам модуля. Например, можно обратиться к константе e, расположенной в модуле math:

```
>>> import math
>>> math.e
2.718281828459045
```

Модуль в языке Python представляет отдельный файл с кодом, который можно повторно использовать в других программах.

Для создания модуля необходимо создать собственно файл с расширением *.py, который будет представлять модуль. Название файла будет представлять название модуля. Затем в этом файле надо определить одну или несколько функций.

Допустим, основной файл программы называется main.py. И мы хотим подключить к нему внешние модули. Для этого сначала определим новый модуль: создадим в той же папке, где находится main.py, новый файл, который назовем message.py. Если используется PyCharm или другая IDE, то оба файла просто помещаются в один проект. Соответственно модуль будет называться message.



```python
hello = "Hello all"


def print_message(text):
    print(f"Message: {text}")
```

Здесь определена переменная hello и функция print_message, которая в качестве параметра получает некоторый текст и выводит его на консоль.

В основном файле программы - **main.py** используем данный модуль:

```python
import message      # подключаем модуль message

# выводим значение переменной hello
print(message.hello)        # Hello all
# обращаемся к функии print_message
message.print_message("Hello work")  # Message: Hello work
```

Для использования модуля его надо импортировать с помощью оператора **import**, после которого указывается имя модуля: `import message`.

Чтобы обращаться к функциональности модуля, нам нужно получить его **пространство имен**. По умолчанию оно будет совпадать с именем модуля, то есть в нашем случае также будет называться **message**.

Получив пространство имен модуля, мы сможем обратиться к его функциям по схеме

пространство_имен.функция

Например, обращение к функции `print_message()` из модуля message:

```
1 message.print_message("Hello work")
```

И после этого мы можем запустить главный скрипт main.py, и он задействует модуль message.py. В частности, консольный вывод будет следующим:

```
Hello all
Message: Hello work
```

Инструкция from

Подключить определенные атрибуты модуля можно с помощью инструкции from. Она имеет несколько форматов:

```
from <Название модуля> import <Атрибут 1> [ as <Псевдоним 1> ], [<Атрибут 2> [ as
 ↪<Псевдоним 2> ] ...]
from <Название модуля> import *
```

Первый формат позволяет подключить из модуля только указанные вами атрибуты. Для длинных имен также можно назначить псевдоним, указав его после ключевого слова as.

```
>>> from math import e, ceil as c
>>> e
2.718281828459045
>>> c(4.6)
5
```

Импортируемые атрибуты можно разместить на нескольких строках, если их много, для лучшей читаемости кода:

```
>>> from math import (sin, cos,
...                   tan, atan)
```

Второй формат инструкции from позволяет подключить все (точнее, почти все) перемен-ные из модуля.

Подключение функциональности модуля в глобальное пространство имен

Другой вариант настройки предполагает импорт функциональности модуля в глобальное пространство имен текущего модуля с помощью ключевого слова **from**:

```
from message import print_message

# обращаемся к функии print_message из модуля message
print_message("Hello work")  # Message: Hello work

# переменная hello из модуля message не доступна, так как она не импортирована
# print(message.hello)
# print(hello)
```

В данном случае мы импортируем из модуля message в глобальное пространство имен функцию `print_message()`. Поэтому мы сможем ее использовать без указания пространства имен модуля как если бы она была определена в этом же файле.

Подключение функциональности модуля в глобальное пространство имен

Все остальные функции, переменные из модуля недоступны (как например, в примере выше переменная hello). Если мы хотим их также использовать, то их можно подключить по отдельности:

```
from message import print_message
from message import hello

# обращаемся к функции print_message из модуля message
print_message("Hello work")  # Message: Hello work

# обращаемся к переменной hello из модуля message
print(hello)    # Hello all
```

Подключение функциональности модуля в глобальное пространство имен

Если необходимо импортировать в глобальное пространство имен весь функционал, то вместо названий отдельных функций и переменных можно использовать символ зводочки *:

```
1  from message import *
2
3  # обращаемся к функции print_message из модуля message
4  print_message("Hello work")  # Message: Hello work
5
6  # обращаемся к переменной hello из модуля message
7  print(hello)    # Hello all
```

Использование псевдонимов

Если название модуля слишком длинное, или оно вам не нравится по каким-то другим причинам, то для него можно создать псевдоним, с помощью ключевого слова as.

```
>>> import math as m
>>> m.e
2.718281828459045
```

Теперь доступ ко всем атрибутам модуля math осуществляется только с помощью переменной m, а переменной math в этой программе уже не будет (если, конечно, вы после этого не напишете import math, тогда модуль будет доступен как под именем m, так и под именем math).

Установка псевдонимов

При импорте модуля и его функциональности мы можем установить для них псевдонимы. Для этого применяется ключевое слово **as**, после которого указывается псевдоним. Например, установим псевдоним для модуля:

```
1  import message as mes  # модуль message проецируется на псевдоним mes
2
3  # выводим значение переменной hello
4  print(mes.hello)        # Hello all
5  # обращаемся к функии print_message
6  mes.print_message("Hello work")  # Message: Hello work
```

В данном случае пространство имен будет называться **mes**, и через этот псевдоним можно обращаться к функциональности модуля.

Установка псевдонимов

Подобным образом можно установить псевдонимы для отдельной функциональности модуля:

```
from message import print_message as display
from message import hello as welcome

print(welcome)          # Hello all - переменная hello из модуля message
display("Hello work")   # Message: Hello work - функция print_message из модуля messa
```

Здесь для функции print_message из модуля message устанавливается псевдоним display, а для переменной hello - псевдоним welcome. И через эти псевдонимы мы сможем к ним обращаться.

Пример создания модуля с функциями для вычисления площадей прямоугольника, треугольника и круга:

Программист на Python всегда может создать собственный модуль, чтобы использовать его в нескольких своих программах.

```
from math import pi, pow

def rectangle(a, b):
    return round(a * b, 2)

def triangle(a, h):
    return round(0.5 * a * h, 2)

def circle(r):
    return round(pi * pow(r, 2), 2)
```

Здесь также иллюстрируется принцип, что один модуль может импортировать другие. В данном случае импортируются функции из модуля **math**.

Создание своего модуля на Python

Создадим файл mymodule.py, в которой определим какие-нибудь функции:

```
def hello():
    print('Hello, world!')

def fib(n):
    a = b = 1
    for i in range(n - 2):
        a, b = b, a + b
    return b
```

Теперь в этой же папке создадим другой файл, например, main.py:

```
import mymodule

mymodule.hello()
print(mymodule.fib(10))
```

Выведет:

```
Hello, world!
55
```

МОДУЛИ СТАНДАРТНОЙ БИБЛИОТЕКИ

Модули стандартной библиотеки условно разбиты на группы по тематике.

Группа	Модули (пакеты)
Сервисы периода выполнения	sys, atexit, copy, traceback, math, cmath, random, time, calendar, datetime, sets, array, struct, itertools, locale, gettext
Поддержка цикла разработки	pdb, hotshot, profile, unittest, pydoc (docutils, distutils)
Взаимодействие с ОС	os, os.path, getopt, glob, popen2, shutil, select, signal, stat, tempfile
Обработка текстов	string, re, StringIO, codecs, difflib, mmap, sgmllib, htmllib, htmlentitydefs (xml)
Многопоточные вычисления	threading, thread, Queue
Хранение данных. Архивация	pickle, shelve, anydbm, gdbm, gzip, zlib, zipfile, bz2, csv, tarfile

МОДУЛИ СТАНДАРТНОЙ БИБЛИОТЕКИ

Группа	Модули (пакеты)
Поддержка сети. Протоколы Internet	cgi, Cookie, urllib, uriparse, httplib, smtplib, poplib, telnetlib, socket, asyncore (SocketServer, BaseHTTPServer, xmlrpclib, asynchat)
Поддержка Internet. Форматы данных	quopri, uu, base64, binhex, binascii, rfc822, mimetools, MimeWdter, multifile, mailbox, (email)
Python о себе	parser, symbol, token, keyword, inspect, tokenize, pyclbr, py_compile, compileall, dis, compiler

Модули могут содержать один или несколько классов, с помощью которых создается объект нужного типа, а затем речь идет уже не об именах из модуля, а об атрибутах этого объекта. И наоборот, некоторые модули содержат общие функции для работы над произвольными объектами.

Пример использования модулей стандартной библиотеки

подключим модуль для работы с временем

и сразу же еще один, для генерации случайных чисел:

```
import time, random

while True:
    print(random.randint(1, 100))
    time.sleep(1)
```

Эта программа будет выводить случайное целое число от 1 до 100 с паузой в одну секунду.

Модуль sys

Модуль sys содержит информацию о среде выполнения программы интерпретатора Python. Ниже представлены наиболее популярные объекты из этого модуля

exit([c]) – выход из программы, можно передать числовой код завершения;
0 – в случае успешного завершения, другие числа при аварийном завершении программы;
argv – список аргументов командной строки. Обычно sys.argv[0] содержит имя запущенной программы, а остальные параметры передаются из командной строки;
platform – платформа, на которой работает интерпретатор
version – версия интерпретатора;
setrecursionlimit(limit) – установка уровня максимальной вложенности рекурсивных вызовов;
exc_info() – информация об обрабатываемом исключении.

Модуль copy

Модуль содержит функции для копирования объектов.

В модуле copy есть еще и функция deepcopy() для глубокого копирования, при которой объекты копируются на всю возможную глубину рекурсивно

Модули math и cmath

В этих модулях собраны математические функции для действительных и комплексных аргументов.

В таблице даны функции модуля math. Там, где аргумент обозначен буквой z, аналогичная функция определена и в модуле cmath.

Функция или константа	Описание
acos(z)	Арккосинус z
asin(z)	Арксинус z
atan(z)	Арктангенс z
atan2(y,x)	atan(y/x)
ceil(x)	Наименьшее целое, большее или равное x
cos(z)	Косинус z
cosh(x)	Гиперболический косинус x
e	Константа e
exp(z)	Экспонента (то есть e * * z)
fabs(x)	Абсолютное значение x
floor(x)	Наибольшее целое, меньшее или равное x
fmod(x, y)	Остаток от деления x на y

Модули math и cmath

frexp(x)	Возвращает мантиссу и порядок x как пару (m, i), где m – число с плавающей точкой, а i – целое, такое что x = m * 2. **i. Если 0 – возвращает (0,0), иначе 0.5<=abs(m)<1.0
hypot(x,y)	sqrt(x*x+y*y)
ldexp(m,i)	m * (2**i)
log(z)	Натуральный логарифм z
log10(z)	Десятичный логарифм z
modf(x)	Возвращает пару (y, q) – дробную и целую часть x. Обе части имеют знак исходного числа
pi	Константа Пи
pow(x,y)	x**y
sin(z)	Синус z
sinh(z)	Гиперболический синус z
sqrt (z)	Корень квадратный от z
tan(z)	Тангенс z
tanh(z)	Гиперболический тангенс z

- factorial(num): факториал числа
- degrees(rad): перевод из радиан в градусы
- radians(grad): перевод из градусов в радианы

Пример применения некоторых функций:

```python
import math

# возведение числа 2 в степень 3
n1 = math.pow(2, 3)
print(n1)  # 8

# ту же самую операцию можно выполнить так
n2 = 2**3
print(n2)

# квадратный корень числа
print(math.sqrt(9))  # 3

# ближайшее наибольшее целое число
print(math.ceil(4.56))  # 5

# ближайшее наименьшее целое число
print(math.floor(4.56))  # 4

# перевод из радиан в градусы
print(math.degrees(3.14159))  # 180
```

```python

# перевод из градусов в радианы
print(math.radians(180))   # 3.1415.....
# косинус
print(math.cos(math.radians(60)))  # 0.5
# синус
print(math.sin(math.radians(90)))   # 1.0
# тангенс
print(math.tan(math.radians(0)))    # 0.0

print(math.log(8,2))    # 3.0
print(math.log10(100))    # 2.0
```

Пример применения некоторых функций:

Также модуль math предоставляет ряд встроенных констант, такие как PI и E:

```
1  import math
2  radius = 30
3  # площадь круга с радиусом 30
4  area = math.pi * math.pow(radius, 2)
5  print(area)
6
7  # натуральный логарифм числа 10
8  number = math.log(10, math.e)
9  print(number)
```

Модуль random

Этот модуль генерирует псевдослучайные числа для нескольких различных распределений. Наиболее используемые функции:

Функция	Описание
random()	Генерирует псевдослучайное число из полуоткрытого диапазона [0.0,1.0)
choice(s)	Выбирает случайный элемент из последовательности s
shuffle(s)	Размешивает элементы изменчивой последовательности s на месте
randrange([start,] stop[, step])	Выдает случайное целое число из диапазона
range (start, stop,step)	Аналогично choice (range (start, stop, step))
normalvariate(mu, sigma)	Число из последовательности нормально распределенных псевдослучайных чисел, mu – среднее, sigma – среднеквадратическое отклонение (sigma > 0)

shuffle(): перемешивает список

Модуль random

В модуле есть функция seed(n), которая позволяет установить генератор случайных чисел в некоторое состояние, например если возникнет необходимость многократного использования одной и той же последовательности псевдослучайных чисел.

Функция **randint(min, max)** возвращает случайное целое число в промежутке между двумя значениями min и max.

```
1  import random
2
3  number = random.randint(20, 35)  # значение от 20 до 35
4  print(number)
```

Модуль random

Функция **random()** возвращает случайное число с плавающей точкой в промежутке от 0.0 до 1.0. Если же нам необходимо число из большего диапазона, скажем от 0 до 100, то мы можем соответственно умножить результат функции random на 100.

```
1  import random
2
3  number = random.random()  # значение от 0.0 до 1.0
4  print(number)
5  number = random.random() * 100  # значение от 0.0 до 100.0
6  print(number)
```

Модуль random

```
1  import random
2
3  number = random.randrange(10)  # значение от 0 до 10 не включая
4  print(number)
5  number = random.randrange(2, 10)  # значение в диапазоне 2, 3, 4, 5, 6, 7, 8, 9
6  print(number)
7  number = random.randrange(2, 10, 2)  # значение в диапазоне 2, 4, 6, 8
8  print(number)
```

Модуль random. Работа со списком

Для работы со списками в модуле random определены две функции: функция **shuffle()** перемешивает список случайным образом, а функция **choice()** возвращает один случайный элемент из списка:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8]
random.shuffle(numbers)
print(numbers)
random_number = random.choice(numbers)
print(random_number)
```

Модуль sets

Модуль реализует тип данных для множеств.

```
import sets
A = sets.Set([1, 2, 3])
B = sets.Set([2, 3, 4])
print A|B, A&B, A-B, A^B
for i in A:
    if i in B:
        print i,
```

В результате будет выведено:

```
Set([1, 2, 3, 4])
Set([2, 3])
Set([1])
Set([1, 4])
2 3
```

Модуль decimal

При работе с числами с плавающей точкой (то есть float) мы сталкиваемся с тем, что в результате вычислений мы получаем не совсем верный результат:

```
1 number = 0.1 + 0.1 + 0.1
2 print(number)       # 0.30000000000000004
```

Проблему может решить использование функции **round()**, которая округлит число. Однако есть и другой способ, который заключается в использовании встроенного модуля **decimal**.

Модуль decimal

Ключевым компонентом для работы с числами в этом модуле является класс **Decimal**. Для его применения нам надо создать его объект с помощью конструктора. В конструктор передается строковое значение, которое представляет число:

```
1  from decimal import Decimal
2
3  number = Decimal("0.1")
```

После этого объект Decimal можно использовать в арифметических операциях:

```
1  from decimal import Decimal
2
3  number = Decimal("0.1")
4  number = number + number + number
5  print(number)        # 0.3
```

Модуль csv

Формат CSV (comma separated values – значения, разделенные запятыми) достаточно популярен для обмена данными между электронными таблицами и базами данных. Следующий ниже пример посвящен записи в CSV-файл и чтению из него:

```
mydata = [(1, 2, 3) , (1, 3, 4)]
import csv
## Запись в файл:
f = file("my.csv", "w")
writer = csv.writer(f)
for row in mydata:
        writer.writerow(row)
f.close()
#Чтение из файла:
reader = csv.reader(file("my.csv"))
for row in reader:
        print row
```

Модуль csv

В данном случае количество рекурсивных вызовов растет экспоненциально от числа n, что совсем не соответствует временной сложности решаемой задачи.

При работе с рекурсивными функциями можно легко превысить глубину допустимой в Python рекурсии. Для настройки глубины рекурсии следует использовать функцию setrecursionlimit(N) из модуля sys, установив требуемое значение N.

Модуль os

Модуль os предоставляет множество функций для работы с операционной системой, причём их поведение, как правило, не зависит от ОС. **Модуль os позволяет взаимодействовать с операционной системой** - узнавать/менять файловую структуру, переменные среды, узнавать имя и права пользователя и др. Программа, использующая переменные и функции модуля os, переносима с одной операционной системы на другую, так как os умеет учитывать особенности каждой ОС. Однако ряд функций используется только для Windows или Unix-подобных ОС.

Следует отметить, что часть функциональности os реализуют другие модули и встроенные функции Python. В этом случае нередко лучше выбирать их. Например, функция os.access() проверяет наличие доступа к файлу. Если файл открывается на чтение или запись, проще использовать функцию open()

Пример некоторых функций модуля os для работы с файловой системой:

Функция	Действие
listdir(dir)	Возвращает список файлов в каталоге dir
mkdir(path[, mode])	Создает каталог path
makedirs(path[, mode])	Аналог mkdir(), создающий все необходимые каталоги, если они не существуют. Возбуждает исключение, когда последний каталог уже существует

Пример некоторых функций модуля os для работы с файловой системой:

Узнать текущий каталог:

```
>>> os.getcwd()
'/home/pl/Documents/python'
```

Создать каталог:

```
>>> os.mkdir('TEXT')
```

Смена текущего каталога:

```
>>> os.chdir('/home')
>>> os.getcwd()
'/home'
>>> os.chdir('./pl/Documents/python')
>>> os.getcwd()
'/home/pl/Documents/python'
```

Создать дерево каталогов:

```
>>> os.makedirs('ONE/TWO/THREE')
>>> os.listdir('ONE')
['TWO']
>>> os.listdir('ONE/TWO')
['THREE']
```