eebb

ЯЗЫК PYTHON

Лекция 15. Модули, пакеты, библиотеки

Мотивация

- Большие задачи надо разбивать на части
- Для этого есть функкции и классы
- Но этого мало
- Нельзя большое приложение держать в одном файле

Мотивация

- Бывает, что в разных приложениях используется общий код
- Нужно как-то обмениваться кодом
- Использовать сторонние библиотеки
- Разрабатывать и распространять свои

Внутри одного проекта

- Начали писать код в одном файле
- Файл стал большим, много кода, глобальных переменных
- Выделим логически связанную часть
- Перенесем в отдельный файл
- В том же каталоге

Модуль math.py

- Вынесем математическую функциональность в отдельный модуль
- Назовем m.py
- math - уже есть стандартный модуль
- Начнем с чисел Фибоначчи
- Опередлим в том же модуле функцию fibo

Использование модуля

- Напишем скрипт main.py
- Импортируем модуль: import m
- Используем функцию: m.fibo
- Убедимся, что работает

Пример: m.py

```
1 def fibo(n):
2     prev, curr = 0, 1
3     for _ in range(n):
4         prev, curr = curr, prev + curr
5 
6     return prev
```

Пример: main.py

```
1 import m
2
3 print(m.fibo(10))
4 print(m.fibo(5))
```

Осознаем происходящее

- Синтаксис - как у вызова метода
- Или обращения к атрибуту объекта
- Не удивительно
- Модуль - это обычный Python-объект
- Как строка, класс, функция

Создание модуля

- Видим import
- Ищем файл с именем модуля и расширением .py
- Создаем объект
- Если нашли файл, исполняем его код

Пример: m.py

```
1 print('before fibo')
2 def fibo(n):
3     prev, curr = 0, 1
4     for _ in range(n):
5         prev, curr = curr, prev + curr
6     return prev
7
8 print('after fibo')
```

Пример: main.py

```
1 print('before import')
2 import m
3 print('after import')
4
5 print(m.fibo(10))
6 print(m.fibo(5))
```

Правила импортирования

- import с одним именем модуля может встречаться много раз
- Возможно, из разных файлов
- Например, основной код импортирует a и b
- И a сам по себе импортирует b

Правила импортирования

- Код верхнего уровня каждого модуля может быть исполнен только один раз
- Если используется стандартный import
- Есть специальная библиотека - может переимпортировать
- Нужна в особых случаях

Модуль двойного назначения

- Бывает так, что код модуля хочется использовать двояко
- Иногда - импортировать, иногда исполнять как программу
- Например, модуля классификации текста
- По тексту определяем - это художественный текст, кулинарный рецепт и т.п.

Модуль двойного назначения

- Как именно - неважно
- Но как-то - можно
- Оформили в классы, функции, сопутствующие исключения
- Можем импортировать и использовать: import classifier

Модуль двойного назначения

- Но можем захотеть исполнить как программу
- python3 classifier.py data.txt
- Чтобы прочитать текст
- И применить к нему логику классификации

Модуль двойного назначения

- Есть встроенная переменная - __name__
- В ней хранится имя модуля
- Если модуль импортирован, в ней хранится имя модуля
- Если файл исполняется как скрипт, в ней хранится '__main__'

Пример: m2.py

```
1 print('before fibo: ', __name__)
2 def fibo(n):
3     print('fibo', __name__)
4     prev, curr = 0, 1
5     for _ in range(n):
6         prev, curr = curr, prev + curr
7     return prev
8
9 print('after fibo', __name__)
```

Пример: main2.py

```
1 print('before import', __name__)
2 import m2
3 print('after import')
4
5 print(m2.fibo(10))
6 print(m2.fibo(5))
```

Модуль двойного назначения

- Можем определить функции, классы и т.п.
- Внизу - if с проверкой равенства __name__ и '__main__'
- Под if - логика на случай приложения
- Разбор аргументов командной строки, вызов логики модуля

Варианты импорта

- Имя модуля становится обычной переменной по месту импортирования
- Это может конфликтовать с именами переменных, функций, классов
- Можно сразу после импорта использовать присваивание
- Но это не очень идиоматично

Пример: main3.py

```
1 import m
2 m2 = m
3
4 value = m2.fibo(10)
5 print(value)
6 print(m2.fibo(5))
```

Варианты импорта

- Более идиоматично - добавить as
- import m as m2
- Ищем модуль m.py, инициализируем как обычно
- Получившийся объект присваиваем сразу в m2

Пример: main4.py

```
1 import m as m2
2
3 value = m2.fibo(10)
4 print(value)
5 print(m2.fibo(5))
```

Варианты импорта

- Иногда не очень хочется использовать префикс
- Чисто технически - можно решить присваиванием
- Но есть встроенная конструкция - from ... import ...
- После from - имя модуля, после import - список имен

Пример: main5.py

```
1 import m
2 fibo = m.fibo
3
4 print(fibo(10))
5 print(fibo(5))
```

Пример: main6.py

```
1 from m import fibo
2
3 print(fibo(10))
4 print(fibo(5))
```

Варианты импорта

- Можно загрузить все символы модумя прямо в текущее пространство
- from ... import *
- В программе - нежелательно
- В силу неконтролируемости
- Удобно в командной строке или как быстрое решение

Пример: main7.py

```
1 from m import *
2
3 print(fibo(10))
4 print(fibo(5))
```

Переменные и контексты

- У каждого модуля - свой глобальный контекст
- Нет возможности из функции модуля увидеть переменную глобального контекста приложения
- Из приложение увидеть глобальную переменную модуля можно
- Как атрибут переменной-модуля

Пример: mvar.py

```
1 V = 123
2
3 def f(value):
4     global V
5     V = value
```

Пример: main-var.py

```
1 import mvar as m
2 V = 234
3
4 m.f(555)
5 print(V)
6 print(m.V)
```

Промежуточный итог

- Умеем выносить общую функциональность в модуль
- И делать модуль двойного назначения
- Но файл привязан к текущему каталогу
- А если функциональность модуля нужна в другом проекте ?
- Который совсем в другом каталоге

Идея решения

- Хранить модули в специальных каталогах
- И как-то указывать Python-у имена этих каталогов
- Но модуль внутри проекта тоже имеет смысл
- В первую очередь Python ищем модуль в том же каталоге, где и скрипт, содержащий import

Идея решения

- В операционных системах есть переменные окружения
- Это механизм настройки
- Можно устанавливать в разные значения
- Приложения смотрят на некоторые переменные окружения
- И настраивают поведение

Идея решения

- Python ориентируется на переменную окружения PYTHONPATH
- Список имен каталогов
- Разделенных двоеточием
- В Windows - точкой с запятой

Идея решения

- Проходит по всем каталогам из списка по очереди
- Пока не найдет в каком-либо файл с именем каталога и расширением .py
- Ее редко приходится править
- Но полезно понимать

Распространение модулей

- Передаем пользователю исходники модуля
- Он их устанавливает в нужный каталог
- Чтобы Python мог их найти
- Вместо "передаем исходники" - скорее даем ссылку для скачивания

Распространение модулей

- Можно автоматизировать
- И организовать хранилище
- Получается инфраструктура распространения пакетов
- Называется pip - Python Installer Package

Распространение модулей

- Утилита pip распространяется вместе с Python
- Хотим установить внешнюю библиотеку - сначала узнаем ее имя в pip
- Запускаем pip install <имя>
- Например, pip install numpy

Разовьем идею

- Пусть есть подзадача, для которой нужно создать модуль
- Например, обработка изображений
- Подзадача сама может декомпозироваться
- Можно обрабатывать разные форматы изображений

Разовьем идею

- А можно по-разному обрабатывать
- Накладывать разные эффекты
- Интерпретировать изображения
- Напрашивается дальнейшее разбиение на модули

Разовьем идею

- Можно сделать несколько отдельных модулей
- Но тут есть минусы
- Нам могут потребоваться внутренние вспомогательные классы
- Например, чтобы представлять изображение независимо от формата файла
- И хочется иерархичности имен

Пакеты

- Есть отдельная конструкция - пакет (package)
- Создаем каталог
- В каталоге - файл __init__.py
- Возможно, пустой
- Наличие такого файла - маркер пакета

Пакеты

- Можно создать несколько py-файлов в каталоге
- Например, каталог image и файлы filters.py, readers.py
- Пусть в filters.py есть функция blur
- А в readers.py - класс JpegReader

Пакеты

- Можно импортировать image.filters или image.readers
- И использовать image.filters.blur или image.readers.JpegReader
- Или импортировать from image import filters, readers
- И использовать filters.blur или readers.JregReader

Пакеты

- Можно уточнить семантику импортирования со звездочкой
- Определив в __init.py__ переменную __all__
- В ней хранится перечень того, что импортирется при звездочке
- Какой-то служебный модуль можно не включать

Пакеты

- Можно использовать несколько вложенных каталогов
- Каждый с файлом __init__.py
- Для больших пакетов со сложной структурой
- Можно импортировать модуль того же пакета
- Используя . или .. после from