Многофайловые проекты

В программах крупнее нескольких сотен строк становится трудно ориентироваться в исходном коде, если он записан в одном файле. По этой причине код разбивается на несколько файлов.

Модули

Каждый файл с расширением .py называется модулем. Если два модуля находятся в одной папке (даже если она не является пакетом), можно осуществлять из них импорт.

```
# class1.py
class Class1:
    pass

# class2.py
import class1  # файл class1.py в той же папке

class Class2:
    def __init__(self):
        self.other = class1.Class1()
```

Если не нравится наличие префикса из имени модуля, можно осуществить from-импорт:

```
# class2.py
from class1 import Class1

class Class2:
    def __init__(self):
        self.other = Class1()
```

Если по той или иной причине имя импортируемой сущности не нравится (например, в этом файле уже есть сущность с таким именем), ее можно переименовать:

```
# class2.py
from class1 import Class1 as C1

class Class1: # сущность Class1 у нас уже есть
    def __init__(self):
        self.other = C1()
```

Обычные импорты тоже можно переименовывать:

```
import numpy as np
```

Особенно это удобно, если имя импорта длинное:

```
import myproject.utils.foobar as fb
```

Пакеты

Программы могут разрастаться до таких размеров, что станет трудно ориентироваться во множестве файлов с исходным кодом в одной папке. В этом случае можно сгруппировать файлы .py в различные подпапки. Чтобы Python мог к ним обращаться, в каждой папке должен находиться файл __init__.py (два подчеркивания с обеих сторон от имени). Этот файл может быть даже пустым. Папка с находящимся в ней файлов __init__.py называется пакетом. Пакеты могут быть и вложенные.

Если модуль находится по пути: Корень_проекта/myproject/utils/foobar.py, то подключить его можно одним из следующих способов:

```
import myproject.utils.foobar
from myproject.utils import foobar
import myproject.utils.foobar as foobar
```

Относительные импорты

Если в пакете myproject.utils находятся два файла: foobar.py и baz.py, то им необязательно обращаться друг к другу от корня проекта. Для этого используется специальное имя пакета **.** (точка). Она обозначает пакет, в котором находится модуль. Таким образом, импортировать foobar.py из baz.py можно так:

```
from . import foobar
# или
from .foobar import some_name
```

А следующее объявление синтаксически некорректно:

```
import .foobar  # так нельзя!
```

Таким образом, относительные импорты это всегда from-импорты.

Имя **..** (две точки) обозначает пакет выше (тот, в котором находится текущий пакет), **...** (три точки) – пакет на два уровня выше и т.д. пока мы находимся внутри системы пакетов. Обратиться так к файлам в папке, в которой нет файла __init__.py нельзя.

```
from .. import xyzzy # файл xyzzy.py на одну папку выше
```

Файл __init__.py

Объявления в этом файле могут быть импортированы напрямую из пакета так, как-будто это не пакет, а модуль.

```
# Корень_проекта/utils/MyClass.py
class MyClass:
    pass

# Корень_проекта/utils/__init__.py
from .MyClass import MyClass

# Корень_проекта/main.py
from utils.MyClass import MyClass
# или
from utils import MyClass # благодаря объявлению в __init__.py
```

Стоит помнить, что относительные импорты вызывают импорт в том числе и из файла __init__.py, поэтому такого рода объявления могут приводить к трудноустранимым циклическим зависимостям.

Циклические зависимости

Нередкой является ситуация, когда два класса ссылаются друг на друга:

```
# class1.py
from class2 import Class2

class Class1:
    def __init__(self):
        self.other = Class2()

# class2.py
from class1 import Class1

class Class2:
    def __init__(self):
        self.other = Class1()
```

В данном случае Python выдаст ошибку о том, что один из классов неопределен. Связано это с тем, что модули будут пытаться импортировать друг друга до тех пор, пока не достигнут максимального уровня вложенности. В этом случае один из классов действительно окажется неопределен.

Для решения данной проблемы стоит использовать обычные импорты, так как в отличие от from-импортов они не пытаются определить содержимое импортированных модулей прямо во время импорта.

```
# class1.py
import class2

class Class1:
    def __init__(self):
        self.other = class2.Class2()

# class2.py
import class1

class Class2:
    def __init__(self):
        self.other = class1.Class1()
```

В данном случае Python не будет жаловаться.