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) переменные из модуля. Для примера импортируем все атрибуты из модуля sys:

```
>>>
>>> from sys import *
>>> version
'3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:03:43) [MSC v.1600 32 bit (Intel)]'
>>> version_info
sys.version_info(major=3, minor=3, micro=2, releaselevel='final', serial=0)
```

Следует заметить, что не все атрибуты будут импортированы. Если в модуле определена переменная __all__ (список атрибутов, которые могут быть подключены), то будут подключены только атрибуты из этого списка. Если переменная __all__ не определена, то будут подключены все атрибуты, не начинающиеся с нижнего подчёркивания. Кроме того, необходимо учитывать, что импортирование всех атрибутов из модуля может нарушить пространство имен главной программы, так как переменные, имеющие одинаковые имена, будут перезаписаны.

При выборе имени модуля помните, что вы (или другие люди) будут его импортировать и использовать в качестве переменной. Модуль нельзя именовать также, как и ключевое слово (их список можно посмотреть тут). Также имена модулей нельзя начинать с цифры. И не стоит называть модуль также, как какую-либо из встроенных функций. То есть, конечно, можно, но это создаст большие неудобства при его последующем использовании.

Выбирая мето расположения модуля, нужно учитывать, что его следует разместить там, где его потом можно будет легко найти. Пути поиска модулей указаны в переменной sys.path. В него включены текущая директория (то есть модуль можно оставить в папке с основной программой), а также директории, в которых установлен python. Кроме того, переменную sys.path можно изменять вручную, что позволяет положить модуль в любое удобное для вас место (главное, не забыть в главной программе модифицировать sys.path).

Модулю в Pythonможно использовать как отдельную программу. Однако надо помнить, что при импортировании модуля его код выполняется полностью, то есть, если программа что-то печатает, то при её импортировании это будет напечатано. Этого можно избежать, если проверять, запущен ли скрипт как программа, или импортирован. Это можно сделать с помощью переменной __name__, которая определена в любой программе, и равна "__main__", если скрипт запущен в качестве главной программы, и имя, если он импортирован.

Оборудование и материалы.

Персональный компьютер, среда разработки Python.

Указания по технике безопасности:

Соответствуют технике безопасности по работе с компьютерной техникой.

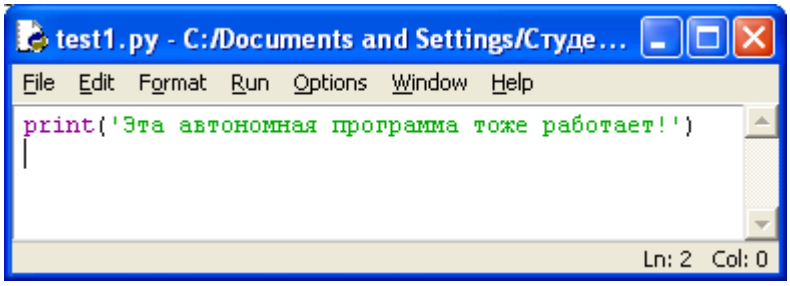
Задания

Отдельные программы

До этого момента вы писали изапускали спомощью интерактивного интерпретатора Python фрагменты кода вроде следующего:

```
>>> print('Строка ввода работает как часы')
Строка ввода работает как часы
```

Теперь создадим вашу первую отдельную программу. Создайте файл под названием test1.py, содержащий следующую строку кода:

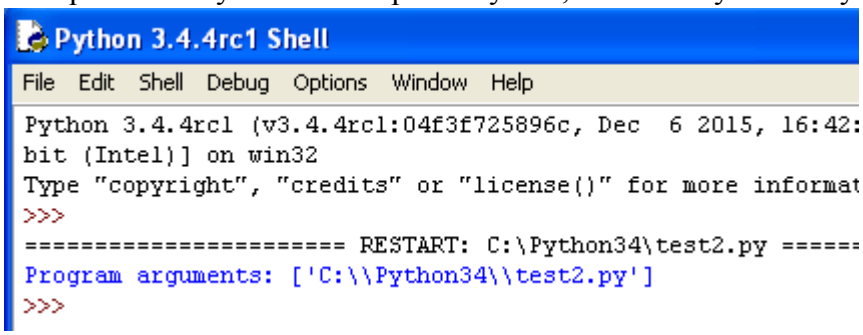


Обратите внимание на отсутствие символов >>>, перед вами лишь одна строка кода. Убедитесь, что перед **print** нет пробелов.

Создайте файл test2.py, который содержит две следующие строки:

```
import sys
print('Program arguments:', sys.argv)
```

Теперь используйте свою версию Python, чтобы запустить эту программу.



Мы собираемся перейти на новый уровень— создание ииспользование кода более чем из одного файла. Модуль— это всего лишь файл, содержащий код Python.

Импортируем модуль

Простейший вариант использования оператора import выглядит как **import Модуль**, где **Модуль**— это имя другого файла Python без расширения .py. Симулируем работу метеостанции и выведем на экран отчет опогоде. Основная программа выведет на экран отчет, аотдельный модуль, содержащий одну функцию, вернет описание погоды, которое будет использовано вотчете.

Основная программа выглядит так (назовем её weatherman.py):

```
import report
description = report.get_description()
print("Today's weather:", description)
```

А ее модуль (report.py) — так:

```
def get_description(): # смотрите строку документации
"""Return random weather, just like the pros"""
from random import choice
possibilities = ['Дождь', 'Снег', 'Дождь со снегом', 'Туман', 'Солнце', 'Кто знает']
return choice(possibilities)
```

Если вы поместите оба этих файла водин каталог иукажете Python запустить файл weatherman.py в качестве основной программы, он обратится к модулю report и запустит его функцию get_description(). Мы написали эту версию функции get_description() так, чтобы она возвращала случайную строку из списка, которую выведет на экран основная программа:

```
>>>
====================== RESTART: C:\Python3
4\weatherman.py ======================
Сегодняшняя погода: Дождь со снегом
>>>
====================== RESTART: C:\Python3
4\weatherman.py ======================
Сегодняшняя погода: Кто знает
>>>
====================== RESTART: C:\Python3
4\weatherman.py ======================
Сегодняшняя погода: Солнце
```

Мы использовали оператор import в двух местах.

- В основной программе weatherman.py, импортируемой модулем report.
- В файле модуля report.py функция get_description() импортирует функцию choice из стандартного модуля Python random.
- Мы также использовали эти операторы двумя разными способами.
- Основная программа делала вызов import report и затем вызывала функцию report.get_description().
- Функцияget_description() измодуляreport.py содержитвызовыfrom random import choice ichoice(possibilities).

В первом случае мы импортировали модуль report целиком, при этом нам нужно было добавить префикс report., чтобы вызвать функцию get_description(). После этого оператора import все содержимое файла report.py становится доступным основной программе, нужно лишь ставить перед именем вызываемой функции префикс report.. Путем уточнения содержимого модуля с помощью его имени мы избегаем возникновения неприятных конфликтов именования. В каком-то другом модуле также может быть функция get_descpirtion(), и мы не вызовем ее по ошибке.

Во втором случае мы находимся внутри функции и знаем, что существует только одна функция с именем choice, поэтому импортируем функцию choice() непосредственно из модуля random. Мы могли бы написать функцию как следующий сниппет, который возвращает случайный результат:

```
def get_description():
    import random
    possibilities = ['Дождь', 'Снег', 'Дождь сл снегом', 'Туман', 'Солнце', 'Кто знает']
    return random.choice(possibilities)
```

Как и для многих других аспектов программирования, выбирайте стиль, который кажется вам наиболее прозрачным. Имя функции, перед которым стоит имя модуля (random.choice), использовать безопаснее, однако из-за этого придется набирать немного больше текста.

Эти примеры применения функции get_description() продемонстрировали варианты того, что можно импортировать, но не показали, где следует выполнять импортирование, — в них import вызывался изнутри функции. Мы могли бы импортировать random из другой функции:

```
>>> import random
>>> def get_description():
...     possibilities = ['rain', 'snow', 'sleet', 'fog', 'sun', 'who knows']
...     return random.choice(possibilities)
...
>>> get_description()
'who knows'
>>> get_description()
'rain'
```

Вам следует рассмотреть возможность импортировать код вне функции, если импортируемый код может быть использован более одного раза, и изнутри функции, если вы знаете, что использование кода будет ограничено. Некоторые люди предпочитают размещать все операторы import в верхней части файла, чтобы явно обозначить все зависимости их кода. Оба варианта работают.

Импортируем модуль с другим именем

В нашей основной программе weatherman.py мы делали вызов import report. Но что, если у вас есть другой модуль с таким же именем или вы хотите использовать более короткое или простое имя? В такой ситуации можете выполнить импорт с помощью псевдонима. Используем псевдоним wr:

```
import report as wr
description = wr.get_description()
print("Today's weather:", description)
```

Импортируем только самое необходимое

С помощью Python вы можете импортировать одну или несколько частей модуля. Каждая часть может сохранить свое оригинальное имя, или же вы можете дать ей alias. Для начала импортируем функцию get_description() из модуля report с помощью его оригинального имени:

```
from report import get_description
description = get_description()
print("Today's weather:", description)
```

Теперь импортируем ее как do_it:

```
from report import get_description as do_it
description = do_it()
print("Today's weather:", description)
```

Каталоги поиска модулей

Где Python ищет файлы для импорта? Он использует список имен каталогов и ZIP-архив, хранящийся в стандартном модуле sys, как переменную path. Вы можете получить доступ к этому списку и изменить его. Вот так выглядит значение переменной sys.path в Python 3.3 одной из версий операционной системы:

```
>>> import sys
>>> for place in sys.path:
...     print(place)
...
/Library/Frameworks/Python.framework/Versions/3.3/lib/python33.zip
/Library/Frameworks/Python.framework/Versions/3.3/lib/python3.3
/Library/Frameworks/Python.framework/Versions/3.3/lib/python3.3/plat-darwin
/Library/Frameworks/Python.framework/Versions/3.3/lib/python3.3/lib-dynload
/Library/Frameworks/Python.framework/Versions/3.3/lib/python3.3/site-packages
```

Пустое место в начале вывода содержит в себе строку '', которая символизирует текущий каталог. Если строка '' находится первой в sys.path, Python сначала выполнит поиск в текущем каталоге, когда вы попробуете что-то импортировать: importreport выглядит как report.py.

Будет использован первый найденный модуль. Это означает, что, если вы определите модуль с именем random и он будет найден раньше оригинального модуля, вы не получите доступ к стандартной библиотеке random.

Контрольные вопросы

1. Понятие модуля.
2. Стандартные и пользовательские модули.
3. Подключение модуля из стандартной библиотеки.
4. Использование функций модуля.
5. Передача аргумента в функцию модуля.
6. Ошибка импорта. Использование псевдонимов.
7. Инструкция from.
8. Выбор имени и места расположения модуля.

Список литературы, рекомендуемый к использованию по данной теме:

1. Орлов, С. А. Программная инженерия. Технологии разработки программного обеспечения : учебник / С.А. Орлов. - 5-е изд., обновл. и доп. - СПб. : Питер, 2017. - 640 с.
2. Гагарина, Л. Г. Современные проблемы информатики и вычислительной техники : [учеб. пособие] / Л.Г. Гагарина, А.А. Петров. - М. : Форум, 2016. - 368 с.
3. Шкляр, М. Ф. Основы научных исследований : учеб. пособие / М.Ф. Шкляр. - 6-е изд. - М. : Дашков и Ко, 2016. - 208 с.
4. Михеева, Е. В. Информационные технологии в профессиональной деятельности : учеб. пособие / Е.В. Михеева. - 14-е изд., стер. - М. : Академия, 2016. - 384 с.
5. Любанович Билл. Простой Python. Современный стиль программирования. — СПб.: Питер, 2016. — 480 с.

Лабораторная работа 10. Сложная обработка текстовых строк

Цель работы:

Форматы кодировок. Строки формата Unicode в Python 3. Модуль unicodedata. Кодирование и декодирование с помощью кодировки UTF-8. Определение кода символа.

Компетенции: