БИБЛИОТЕКИ (МОДУЛИ)

подключение готовых, создание собственных

Библиотека (или модуль) — набор методов (функций) в рамках определенной темы. Уже знакомая нам библиотека ***turtle*** является подключаемым модулем для работы с векторной («черепашьей») графикой.

Используя возможности данной библио теки, мы можем создавать различные узоры.

Подключение библиотеки осуществляется с помощью ключевого слова (команды) ***import*** (при необходимости, в сочетании с такими ключевыми словами как ***as*** и ***from***).

Например, подключение функции (мето да), «перемешивания» списка из библиотеки ***random***, будет выглядеть так:

```
from random import shuffle
# создаем список чисел: от 0 до 9
nums = list(range(10))
print('Исходный список:', ', '.join(map(str, nums)) +
'.')
shuffle(nums) # "перемешиваем" значения
print('После "перемешивания":',
      ', '.join(map(str, nums)) + '.')
```

БИБЛИОТЕКИ (МОДУЛИ)

подключение готовых, создание собственных

Подключая модуль ***random*** привычным способом, можно записать так:

```
import random as r

# создаем список чисел: от 0 до 9
nums = list(range(10))

print('Наш список:', ', '.join(map(str, nums)) + '.')

r.shuffle(nums) # "перемешиваем" значения

print('Случайный выбор:', str(r.choice(nums)) + '.')
```

Вспомним модуль ***datetime*** и рассчита ем время пребывания в командировке:

```
import datetime as dt

# дата и время отъезда
leave_time = dt.datetime(2020, 8, 10, 9, 0, 0)

# дата и время приезда
arrival_time= dt.datetime(2020, 8, 12, 17, 00, 0)

print('Уехал:', leave_time)
print('Приехал:', arrival_time)

# затраченное время
elapsed_time = arrival_time - leave_time
print('Провёл в командировке:', elapsed_time)
```

Полученный результат представлен в международном формате.

Для того, чтобы использовать привычные для нас названия дней и формат дат, зададим местоположение с помощью библиотеки ***locale***:

```
import datetime as dt
import locale # подключение для локализации

# задаем местоположение - RU
locale.setlocale(locale.LC_ALL, "ru")

# дата и время отъезда
leave_time = dt.datetime(2020, 8, 10, 9, 0, 0)

# дата и время приезда
arrival_time= dt.datetime(2020, 8, 12, 17, 00, 0)

print('Уехал:', leave_time.strftime('%a, %d %b.,\
%Y г., в %H:%M'))

print('Вернулся:', arrival_time.strftime('%a, %d \
%b., %Y г., в %H:%M'))

# затраченное время
elapsed_time = arrival_time - leave_time

print('Провёл в командировке:',
elapsed_time.days, 'дн. и', elapsed_time.seconds //
3600, 'ч.',)
```

БИБЛИОТЕКИ (МОДУЛИ)

подключение готовых, создание собственных

В языке Python есть возможность создавать и подключать собственные модули. Для этого достаточно сохранить в отдельный файл набор полезных функций. Например, пусть это будет файл, с именем mylib.py

```
# создаем библиотеку (модуль mylib.py)
def greeting(name):
    if isinstance(name, str):
        return 'Привет, ' + name + '.'
    else:
        return 'Привет, ' + str(name) + '.'
```

Этот файл (mylib.py) нужно поместить в тот же каталог папку, где будет помещен файл с кодом для его вызова.

Вызов будет выглядеть так (файл libtest.py):

```
# libtest.py вызывает функцию модуля mylib.py
import mylib as ml
print(ml.greeting('Алексей')) #вызов функции
```

Разумеется файл (mylib.py) может хранить намного больше полезных функций.

БИБЛИОТЕКИ (МОДУЛИ)

подключение готовых, создание собственных

Модуль mylib был подключен без особого труда, так как находился в том же каталоге (папке), что и исполняемый скрипт.

Существует возможность подключать модули с расположением в других папках на диске персонального компьютера (ПК). Если подключаемый модуль не был найден в папке, из которой запускается исполняемый скрипт (в нашем случае – libtest.py), то интерпретатор будет искать также в консольной переменной PYTHONPATH. Чтобы узнать, где именно будет осуществлён поиск на конкретном ПК,

```
import sys  # подключаем библиотеку
            # для работы с системой
sys.path #список всей путей для библиотек
```

В ответ будет выведен список всех путей, прописанных в указанной выше переменной PYTHONPATH. Среди них первый, как можно заметить, как раз и есть папка, откуда запускается скрипт.

БИБЛИОТЕКИ (МОДУЛИ)

подключение готовых, создание собственных

Если необходимо уточнить, какие стандартные библиотеки возможны для использования (подключения) необходимо однострочный скрипт:

```
help('modules')
```

В ответ получим сообщение:

Please wait a moment while I gather
a list of all available modules...

и через несколько секунд будет выведен список для импорта (**import**) и работы с библиотекой (библиотеками).

Для того, чтобы уточнить все имена функций (и не только) в подключаемом модуле, используется функция ***dir***().

```
# список имен функций до подключения mylib.py
print(dir()) # выводим их
import mylib as ml
print(dir()) # теперь выводим с учётом mylib.py
# print(dir(ml)) # или так
print(ml.greeting('Алексей')) #вызов функции
```

Задача: узнать перечень методов ***turtle***.

БИБЛИОТЕКИ (МОДУЛИ)

подключение готовых, создание собственных

Разбивая программу на разные файлы, логические разделяя на блоки, упрощает её дальнейшее редактирование и поддержку.

Планировать указанное выше разделение рекомендуется ещё до начала написания программы, определив как, где и за что будет отвечать отдельно взятый блок (модуль).

Поскольку каждый отдельный модуль – это файл, с расширением «**.py**», важно определить какой из них будет *подключаемым*, а какой *исполняемым*, т.е.

```
# функция для приветствия по имени
def greeting(name):
    return f'Привет,  {name}.'

if __name__ == '__main__':
    print(f'Это исполняемый, главный\
{__name__} модуль.')
else:
    print(f'Это подключаемый {__name__}
модуль.')
```

Теперь если вызвать его как подключае мый модуль (с помощью import):

```
# libtest.py вызывает функцию модуля mylib.py
import mylib as ml
print(ml.greeting('Алексей')) #вызов функции
```

то можно увидеть следующий результат:

```
Это подключаемый mylib модуль.
Привет,  Алексей.
>>>
```

Теперь стали понятными, как минимум, три важных момента:

1. Подключаемый модуль выполняется полностью. При необходимости использовать только часть его функций, необходимо вызывать их директивой **from**.

2. Все, что находится вне функций (т.е. глобально), выполнится в любом случае.

3. В исполняемом (запускаемом) модуле глобальная переменная ***__name__*** получает значение **__main__**.

БИБЛИОТЕКИ (МОДУЛИ)

подключение готовых, создание собственных

Следовательно, при программе, состоящей из основного (исполняемого) и подключаемых модулей, конструкция:

```
if __name__ == '__main__':
    # команды, вызовы функций основного модуля
    # в т.ч. из подключаемых модулей
```

позволяет как назначить основной модуль, так и предотвратить некорректные результаты при попытке запуска модуля, предназначенного только для подключения.

Не все библиотеки устанавливаются вместе с интерпретатором. Некоторые приходится устанавливать дополнительно. Для этого пользуются менеджером пакетов **PIP**.

Начиная с Python версии 3.4, pip поставляется вместе с интерпретатором. Запускается из командной строки системы Windows, вызовом **Win+R** и далее командой **cmd**. Синтакис:

pip install pygame

В примере выше устанавливается библиотека **pygame** (устанавливать данную библиотеку пока не обязательно).

подключение готовых, создание собственных

ИТОГ УРОКА

1. Модули в Python – это отдельные файлы с расширением **.py**. Имя модуля для ис пользования и есть имя файла (без рас ширения).
2. Модуль Python может иметь набор опре деленных функций, структур данных или переменных.
3. Модули импортируются с помощью ко манды **import**, а также вспомогательных директив: **as** и **from**.
4. Импортируемый модуль выполняется полностью.
5. Для определения исполняемого модуля используется глобальная переменная __name__, которая (для основного модуля) принимает значение __main__. В противном случае её значение – имя модуля.
6. Благодаря модулям (библиотекам) Python является расширяемым языком. Множество полезных библиотек можно установить с помощью менеджера пакетов PIP.

БИБЛИОТЕКИ (МОДУЛИ)

подключение готовых, создание собственных

ДОМАШНЕЕ ЗАДАНИЕ

Даны два модуля:

1. f_lib.py – библиотека с функцией multiply:

```
#f_lib.py - функция использует аннотацию:

# l: list - значит, что аргумент - это список
# -> - значит, что функция вернет тип int

def multiply(l: list) -> int:
    length = len(l) # длина списка
    n = 1 # объявляем переменную
    for i in range(length):
        n *= l[i] # перемножает все элементы списка
    return n

# написать здесь условие, которое при запуске
# выведет: "Это не основной модуль!"
```

2. main_module.py – подключаемый модуль, для вызова функции multiply:

```
#main_module.py

int_list = [1, 2, 3, 4] # список из 4 элементов
                        # типа int

# написать здесь условие, которое при запуске
# вызовет библиотеку (модуль) f_lib.py и с помощью
# функции multiply перемножит все элементы int_list.
```

СПРАВОЧНИК
Часть VI

Кортежи:

```
tuple1 = (1, 2, 3) # классический способ
tuple2 = 1, 2, 3    # без скобок можно,
                    #  но не рекомендуется
tuple3 = (1,) # одно значение с запятой
tuple4 = 1,   # значение с запятой → тоже
# кортеж (без скобок не рекомендуется)
tuple5 = tuple('кортеж') # функцией tuple
```

Операции:

```
tuple1 = (1, 2, 3)
tuple1 = (4, 5, 6)
# сумма кортежей
num_tuple = tuple1 + tuple2
num_tuple += (7,)
#(1,2,3,4,5,6,7)
length = len(num_tuple) # длина
print(num_tuple[0]) – item
```

Множества и их отличие от списков и кортежей:

```
num_set = set() # пустое множество
num_set = set([1, 2, 3, 2]) # или из списка
# из строк, с помощью фигурных скобок
str_set = {'один', 'два', 'три', 'два'}
# структура данных не упорядочена, но исключает повторы
```

```
# список с повтором значений
num_list = [3, 1, 4, 3, 2, 1, 3]
# set исключает повторы
num_set = set(num_list)
# сортируем, сохраняя в список
num_list = sorted(num_set)
```

Операции с множествами:

```
C = A | B # или C = A.union(B) – объединение, с исключением повторов
C = A & B # или A.intersection(B) – пересечение (повторяющиеся элементы)
C = A - B # или A.difference(B) – элементы, входящие в A, не входящие в B
C = A ^ B # или A.symmetric_difference(B) – не входящие в A и B одновременно
```

Форматирование строк:

```
name = 'Виктор'
height = 141.5

f_string = 'Имя: %18s, рост: %6.1f см.' % (name, height) # place holders
f_string = 'Имя: {0}, рост: {1}.'.format(name, height) # format позиционно
f_string = 'Имя: {:s}, рост: {:.1f}.'.format(name, height) # format & place holders
f_string = f'Имя: {name}, рост: {height} см.' # f – строка
```

Подключаемые модули:

```
import turtle as t # импортирование библиотеки turtle в объект t
from turtle import * # импортирование всех методов (и атрибутов) библиотеки turtle
from random import (shuffle, randint) # импортирование двух методов из random

# проверка, модуль исполняемый или подключаемый
if __name__ == '__main__': # если __main__, то исполняемый
```