*Тема 2.1. Работа с модулями.* ***Лекционное занятие (1 час)***

Модулем в Python может быть любой программный файл, содержащий код, включая функции, процедуры, классы или переменные. Иначе говоря, *любой файл с кодом с расширением(.py), является модулем.* Каждая программа может импортировать модуль и получить доступ к его компонентам. Нужно заметить, что ***модуль может быть написан не только на Python, но и на других языках программирования.***

Использование модулей в программе дает следующие преимущества:

* обеспечивает многократное использование программного кода в различных программах за счет сохранения его в отдельных файлах;
* разбивает поле имен программы на отдельные пакеты, минимизируя вероятность их конфликта, поскольку имена модуля видны в других только если их импортировали;
* даёт возможность раздельной разработки компонентов, а также реализации техники совместного использования модуля импортированием многими клиентами.

Модули в Python можно разделить на четыре основные категории:

|  |  |  |
| --- | --- | --- |
| 1. встроенные; | 2. стандартные библиотечные; | 3. пользовательские; |
| 4. сторонние, присоединенные, например, удаленно, загрузкой из сайта Python. |

**Встроенные модули** представляют базовые возможности языка и либо импортируются автоматически, либо через оператор импорта без необходимости установки.

**Стандартные библиотеки** – набор модулей и пакетов для расширения возможностей языка, входящий непосредственно в состав дистрибутива Python и также требует лишь простого импортирования ее модулей без необходимости дополнительной установки.

**Сторонние модули** – более 90 000 программ, которых нет в дистрибутиве Python, но их можно загрузить из сайта [Python Package Index](https://pypi.org/) (PyPI) с помощью утилиты pip.

**Пользовательские модули** – модули, созданные разработчиками.

Функции в модуле, в отличие от процедур, содержат оператор **return**, указывающий возвращаемое значение, тогда как *процедуры не возвращают ничего*.

Доступ к ресурсам модулей можно получить несколькими способами.

Первый вариант – использовать инструкцию **import**. Общий вид команды:

**import** <имя\_модуля>

Например: **import** os

Одной инструкцией можно подключить несколько модулей, хотя этого не рекомендуется делать когда это снижает читаемость или комментируемость кода.

**import** time, random

Рассмотрим второй вариант. Вместо импортирования всего модуля с помощью **from-import** оператора можно взять лишь отдельные его компоненты:

**from** < имя\_модуля > **import** <name 1>, <name 2>..,<name n>

Например: **from** calculation **import** summation

Можно импортировать и все ресурсы модуля: **from** < имя\_модуля > **import** \*

Если название модуля вам не нравится, то для него можно создать псевдоним, с помощью ключевого слова **as: import** <имя\_модуля> **as** <короткое\_имя\_модуля>

Например: **import** math **as** m

Следует избегать инструкций **import** в теле функции, так как это может приводить к многократному ресурсоемому импортирования при каждом ее вызове в виду очистки памяти после выхода из нее, а потому модуль по умолчанию подгружается только один раз за все время работы программы. При повторных попытках импорта интерпретатор по умолчанию просто использует модуль, уже загруженный в память.

Далее перечислим основные встроенные модули (библиотеки).

Таблица 1. Основные встроенные модули

|  |  |
| --- | --- |
| **Имя модуля** | **Описание** |
| math | Здесь функции вещественных и комплексных аргументов. |
| random | Здесь псевдослучайные числа для различных распределений. |
| time | Здесь функции текущего времени и его форматов. |
| itertools | Итераторы – для последовательной обработки данных без цикла. |
| turtle | Модуль для реализации черепашьей графики. |

Процесс создания модуля ничем не отличается от создания обычной программы, так как любая программа на Python является и модулем. Приведем пример такого модуля – создадим файл m1.py, в котором определим какую-нибудь функцию:

|  |  |
| --- | --- |
| Рис.1. Вид файла m1.py: | **def** pr(name): **print**("Привет", name) |

Создадим файл u\_m1.py, который при запуске обратится к внешней функции m1.pr:

|  |  |  |
| --- | --- | --- |
| **import** m1**print**(m1.pr("Петя")) | Рис.2. Результат работы программы: |  |

Точка выделяет элемент модуля, а **None** – результат обращения к процедуре как к функции внутри команды **print** (в процедуре, в отличие от функции, нет **return**) .

***2.1.Практическое занятие (1 час)***

1. Напишите программу, к которой необходимо подключить модуль math. С помощью ресурсов модуля вычислите следующие математические выражения:

|  |  |
| --- | --- |
| 1. n!, n=6, n=10, n=120. Пример программы:
 | **from** math **import** factorial **as** F**print**(**F**(6), **F**(10), **F**(120), sep="\n") |
| 1. $e^{2x}+log\_{3}x.$ Пример программы:
 | **from** math **import** \*x=**float**(**input**()); **print**(**exp**(2\*x)+**log**(3,x)) |
| 1. $C\_{5}^{2}.$ Пример программы:
 | **from** math **import** \*; **print**(**comb**(5, 2)) |

1. С помощью модуля itertools вычислить $A∙B (A=\left\{1,2,3\right\}, B=^{'}12^{'}).$

|  |  |
| --- | --- |
| Пример: | **from** itertools **import** \*; A = [1, 2, 3]; B = "12"; **print**(\***product**(A, B)) |

1. Напишите программу и, подключив модуль random, получите:
2. Случайное целое число из диапазона [1, 5000]. Пример программы:

**from** random **import** randint as rnd; **print**(**rnd**(1,5000)) #можно писать через «;»

1. Случайное вещественное число из диапазона [0.0, 1.0). Пример программы:

**from** random **import** random as rnd; **print**(**rnd**())

1. Случайное число из заданной последовательности. Пример программы:

**from** random **import** \*; s = [10, 11, 12, 13, 14, 15]; **print**(**choice**(s))

1. Напишите программу и, подключив модуль turtle, получите:1) Квадрат со стороной 20. Пример программы:

**from** turtle **import \***; **setposition**(0, 0); **pd**(); **left**(90); **forward**(20);

**left**(90); **forward**(20); **left**(90); **forward**(20); **left**(90); **forward**(20); **pu**()

2)Треугольник со стороной 60. Пример программы:

**from** turtle **import \*; setposition**(50, 50); **pd**()

**left**(120); **forward**(60); **left**(120); **forward**(60); **left**(120); **forward**(60); **pu**()

3)Букву S. Пример программы:

**from** turtle **import** \*; **setposition**(100, 100); **pd**()

**forward**(50); **left**(90); **forward**(50); **left**(90)

**forward**(50); **right**(90); **forward**(50); **right**(90); **forward**(50); **pu**()

(Можно все 3 рисунка задания 4 выполнять вместе, импоритруя turtle один раз).

*Тема 2.2. Работа с модулями.* ***Лекционное занятие (1 час)***

Пакет – способ структуризации модулей. Это папка, содержащая модули и другие пакеты и, а также файл **\_\_init\_\_.py** инициализации пакета (не обязателен с версии 3.3).

Пакеты создают программисту среду разработки приложений, предоставляя нужную иерархию каталогов, где пакет содержит подпакеты, модули и подмодули. Пакеты используются для эффективной категоризации кода уровня приложения.

Подключение пакета аналогично подключению модуля, которое было рассмотрено в предыдущей лекции. Кроме того, имеется возможность импорта отдельных модулей из пакета, если нет необходимости использовать весь пакет:

|  |  |
| --- | --- |
| **import** <имя\_пакета>**import** <имя\_пакета>.<имя\_модуля1> | Пример: **import** package\_1 Пример: **import** package\_1.module\_1 |

Просмотр содержимого модуля/пакета + справки к нему - функциями **dir**() и [**help**()](https://www.yuripetrov.ru/edu/python/ch_02_02.html#help):

Например, рассмотрим содержимое модуля itertools:



Рис.1. Содержимое модуля **itertools**

Заметим, что содержимое *инициализирующего* модуля и определяет атрибуты объекта пакета. Прочие модули пакета и вложенные пакеты не импортируются с пакетом-родителем, но их можно импортировать отдельно, указав полное имя. Важный момент: импортируя вложенный модуль сначала импортируют модули инициализации родительских пакетов. Каждый модуль имеет *специальные* и *дополнительные* атрибуты.

*Специальные –* путь запуска, имя, и др. - доступны всегда. Рассмотрим некоторые:

**\_\_name\_\_ –** Полное имя модуля. Пример: "math" или "os.path".

**\_\_doc\_\_ –** Строка документации.

**\_\_file\_\_ –** Полный путь к файлу, из которого модуль был создан (загружен).

Пример: C:\code\task\_09\_01\_02\fibonacci.py.

Дополнительные атрибуты - информация об авторе, версии модуля и т.д.:

**\_\_author\_\_**, **\_\_copyright\_\_**, **\_\_credits\_\_**, **\_\_license\_\_**, **\_\_version\_\_**, **\_\_maintainer\_\_**, **\_\_email\_\_**, **\_\_status\_\_**

Рассмотрим пример создания пакета.

Как было сказано ранее, для работы с простым пакетом нужно создать папку, в которую желательно (до версии 3.3 – обязательно) поместить файл **\_\_init\_\_.py**.

Создадим каталог **pac**, в который поместим 2 файла: **fib.py** (это файл модуля, в котором содержится функция вычисления n-го числа Фибоначчи) и файл **\_\_init\_\_.py** (этот файл оставим пустым). Данная структура показана на рис.2.

|  |  |
| --- | --- |
| Рис.2. Структура каталога **pac**: |  |

Также создадим файл test.py со следующим листингом:

**import** pac.fib; **print**(pac.fib.**get\_fib**(3))

Можно задать в файле **\_\_init\_\_.py** следующую строку, подключающую модуль fib:

**from** pac.fib **import** get\_fib

Тогда в файле **test.py** листинг может выглядеть следующим образом:

**import** pac; **print**(pac.**get\_fib**(3))

Полезен также атрибут **\_\_all\_\_**, который в модуле инициализации пакета ведёт себя так же, как и в обычном модуле, но если при импорте пакета "со звёздочкой" среди перечисленных имён встретится имя вложенного модуля, а модуль не импортирован ранее в том же **\_\_init\_\_.py**, то он все равно импортируется, т.е. импортируется неявно!

В Python возможна установка внешних пакетов/модулей, а самые популярные из них есть в **PyPI** (Python Package Index), или же на **github**, **bitbucket**, или **Google Code**.

Существуют четыре варианта установки внешних пакетов:

|  |  |  |  |
| --- | --- | --- | --- |
| * установка из источника;
 | * easy\_install;
 | * pip;
 | – другие способы. |

***2.2.Практическое занятие (1 час)***

1. Напишите модуль поиска длины окружности, площади круга и подключите их.

|  |  |
| --- | --- |
| Примерный вид модуля (u\_circ): | **from** math **import** pi**def** circ**\_**len(r): **return** 2 \* pi \* r**def** circ\_sq(r): **return** pi\*r\*r |
| Примерный вид программы, в которую подключены ресурсы данного модуля: | **import** u\_circ; r=**float**(**input**())**print**("длина окр.:", u\_circ.**circ\_len**(r))**print**("площад круга", u\_circ.**circ\_sq**(r)) |

1. Напишите модуль подсчета периметра треугольника и проверки его существования. Подключите его к программе.

|  |  |
| --- | --- |
| Пример модуля (u\_tr): | **def** tr\_check(a,b,c): **return** a+b>c **and** a+c>b **and** b+c>a**def** tr\_per(a,b,c): **return** a+b+c |
| Пример подключения модуля: | **from** u\_tr **import** \*; a,b,c=**map**(**float**, **input**("a,b,c=").**split**())**if** tr\_check(a,b,c): **print**(tr\_per(a,b,c)) |

1. Напишите модуль c проверкой строки на палиндромом и подключите его.

|  |  |
| --- | --- |
| Пример модуля (u\_pal)два варианта – компактный и быстрый: | **def** pal\_check(s): **return** s==s[::-1]#медленный алгоритм**def** pal\_check(s): #быстрый алгоритм (в 8 раз быстрее) **for** k **in range**(**len**(s)//2): **if** s[k]!=s[**len**(s)-k-1]: **return** False **return** True |
| Пример программы: | **from** u\_pal **import** \*; s=""**if** !**pal\_check**(**input**()): s="не "**print**("это", s, "палиндром") |

1. Напишите модуль с функцией вывода первых n строк треугольника Паскаля:

|  |  |
| --- | --- |
| Вид треугольника Паскаля: |  |
| Примерный вид модуля (u\_pasc): | **def** pasc(n): trow = [1]; y = [0] **for** x **in** **range**(**max**(n,0)): **print**(trow) trow=[l+r **for** l,r **in** **zip**(trow+y, y+trow)] **return** n>=1 |
| Примерный вид программы: | **import** u\_pasc; n=**int**(**input**()); u\_pasc.**pasc**(n) |

***2.3.Практическое занятие (2 часа)***

1. Напишите модуль numbers.py для НОД, НОК и проверки числа на «простоту».

|  |  |
| --- | --- |
| Пример модуля (numbers): | **def** nod(a,b): **while** b!=0: a, b=b, a%b **return** a**def** nok(a,b): **return** a\*b/**nod**(a,b)**def** prost(n): i=2; f=**True** **while** (i\*i<=n) **and** f: **if** n%i==0: f=**False** i+=1 **return** f |
| Пример программы: | **import** numbers; **print**(numbers.**nod**(6,8)) |

1. Напишите модуль word\_utils.py, возвращающий список слов из предложения.

|  |  |
| --- | --- |
| Примерный вид модуля (word\_utils): | **def** cl(+s, p): sn='' **for** x **in** s: **if** x **not** **in** p: sn+=x **else**: sn+=' ' **return** sn**def** words(s): **return** **cl**(s, '.,:;!?').**split**() |
| Примерный вид программы, где подключены ресурсы модуля: | **from** word\_utils **import** \***print**(**words**('dg,ghj.ssg,h56,df.j!hkj? h:d;h')) |

1. Напишите модули triangle, circle и square, которые вычисляют площадь и периметр фигур: треугольник, окружность и квадрат соответственно.

|  |  |
| --- | --- |
| Примерный вид модулей **triangle**, **circle** и **square+tr\_check**: | **from** math **import** \***def** t\_per(a,b,c):  **if** tr\_check(a,b,c): **return** a+b+c**def** tr\_check(a,b,c): **if** a+b>c **and** a+c>b **and** b+c>a: **return** **True** **else**: **return** **False****def** t\_sq(a,b,c): p=(a+b+c)/2; **return** (p\*(p-a)\*(p-b)\*(p-c))\*\*0.5**def** c\_len(r): **return** 2\*pi\*r**def** c\_sq(r): **return** pi\*r\*r**def** s\_per(a): **return** 4\*a**def** s\_sq(a): **return** a\*\*2 |
| Примерный вид программы, где подключены ресурсы **circle**: | **from** circles **import** \***print**(**c\_len**(5))#длина окружности радиуса 5 |

1. Создать *пакет* модулей задания 3 (переносим в **pac1,** создав пустой **\_\_init\_\_.py**.

***2.4.Практическое занятие (2 часа)***

1. Составьте модуль vector.py для: создания структур-вектора; вычисления длины; умножения вектора на число. Пример решения:

|  |  |
| --- | --- |
| Примерный вид модуля vector.py: | **from** random **import** randint**def** vec\_len(s): **return** **len**(s)**def** vec\_new(n): **return** [**randint**(1, 50) **for** i **in** **range**(n)]**def** vec\_mul(s,k): **return** [ k\*x **for** x **in** s] |
| Примерный вид программы: | **from** vector **import** \*;s=[1,2,3]; **print**(**vec\_mul**(s,3)) |

1. Реализуйте модуль nwords1.py, принимающий последовательность слов, через дефис, возвращая строку слов, разделенной дефисами, в отсортрованном виде.

|  |  |
| --- | --- |
| Пример модуля nwords1.py: | **def** nw(s): s=s.**split**('-'); s.**sort**(); **return** '-'.**join**(s) |
| Примерный вид программы: | **from** nwords1 **import** \*; **print**(**nw**('xyz-abc-peq')) |

1. Реализуйте модуль nwords2.py, позволяющий: принимать последовательность слов, через пробел, возвращая строку, где в словах обратный порядок букв.

|  |  |
| --- | --- |
| Примерный вид nwords2.py: | **def** nw(s): sn=s.**split**(' '); sn1=[] **for** x **in** sn: sn1+=[x[::-1]] **return** sn1 |
| Примерный вид программы: | **from** nwords2 **import** \*; s='xyz abcd'; **print**(**nw**(s)) |

1. Реализуйте модуль nwords3.py, позволяющий: принимать как входные данные последовательность слов, разделенных пробелами, возвращать как результат строку, где слова зашифрованы методом «бутерброда»: сначала буквы слова нумеруются в таком порядке: первая получает номер 1, последняя — номер 2, вторая – номер 3, предпоследняя – номер 4, потом третья… и так для всех букв.

|  |  |
| --- | --- |
| Пример модуля nwords3.py: | **def** sh(s): sn=[]; d=**len**(s)//2+ **len**(s)%2 **for** i **in** **range**(d):  sn+=[s[i]]; k=**len**(s)-i-1 **if** i!=k: sn+=[s[k]] **return** ''.join(sn)**def** nw(s): sn=s.**split**(' ') **for** i **in** **range**(**len**(sn)): sn[i]=**sh**(sn[i]) **return** sn |
| Пример программы: | **from** nwords3 **import** \*; **print**(**nw**('hello world 123 4567')) |

1. Объединить три модуля **nwords** в пакет (см. задание 4 предыдущей практики).

*Предлагаемые примеры решений не «оптимальны» с точки зрения краткости записи кода, но являются таковыми с точки зрения быстродействия, памяти и читаемости.*

**Важно!!!** При сдаче заданий на проверку необходимо создавать архив со всеми созданными модулями и папками. Архиватор – любой, но формат – ZIP.

 На следующей странице представлен материал для упрощения отладки программ.

***Материал для «продвинутых»: примеры разработки полезных модулей***

 При отладке программ часто возникает необходимость генерации массивов данных с определенными свойствами. В данном разделе – текст генераторов массивов случайных чисел, символов и строк, а также дан пример их использования.

Текст модуля **arrrnd.py:**

**from** random **import** \*

Eng='qwertyuiopasdfghjklzxcvbnm'; Rus='ёйцукенгшщзхъфывапролджэячсмитьбю'

**def** arrrndint(n,mn,mx): **return** [**randint**(mn,mx) **for** i **in** **range**(n)]#случайные int

**def** arrrndfloat(n,mn,mx): **return** [**uniform**(mn,mx) **for** i **in** **range**(n)] #случайные float

**def** arrrndChr(n,L): **return** [L[**randint**(0, **len**(L)-1)] **for** i **in** **range**(n)] #случайные Chr

**def** arrrndW(n,mn,mx,L):**return** [''.**join**(arrrndChr(**randint**(mn,mx),L))**for** i **in** **range**(n)

Текст модуля **rnd\_tst.py** (пример – отсотрированный частотник символов строки)**:**

**from** arrrnd **import** arrrndW, Rus, Eng# генератор случайных слов и алфавиты

x=**arrrndW**(1,80,80,Rus)[0]#массив:строка из 80 rnd-символов-из массива слов

**print**(x)#вывод сгенерированной строки

z=[[**chr**(i),0] **for** i **in** **range**(65536)] #массив из 65536 пар [Сhr, 0(счетчик повторов)]

**for** c **in** x: z[**ord**(c)][1]+=1 #из Chr->код (индекс в z), увеличиваем счетчик на 1

z.**sort**(key=**lambda** x: (-x[1],x[0].**lower**()))#сортируем одновременно по двум полям:

#1)сначала счетчик со знаком минус (т.е. получим сортировку частот по убыванию)

#2)при равных частотах - по алфавиту Chr без зависимости от регистра (.lower())

**for** i **in** **range**(**len**(z)):

 **if** z[i][1]==0:z=z[:i];**break** #когда остались только нули – обрезаем массив

**print**(\*z, sep='\n')#выводи в столбик символы с их частотами

Отметьте использование генератора массива Chr в генераторе слов+сортировки!!!