

## Раздаточный материал №1

<pre> 1  from tkinter import * 2  from tkinter import ttk 3 </pre>	<p>В результате импортирования в пространстве имён программы (скрипта) появляются имена, встроенные в tkinter, к которым можно обращаться непосредственно. Массовое импортирование имён может привести к их конфликту. Кроме того, для интерпретатора требуется больше времени, чтобы в списке доступных имён найти нужное.</p> <p><b>Основные виджеты- Раздаточный материал № 2</b></p> <p>Подмодуль ttk предоставляет доступ к множеству стилизуемых виджетов tk, а так же предоставляет дополнительные виджеты. Импорт ttk должен следовать за импортом tk</p>
<pre> 4  root = Tk() 5  root.title("Привет мир!") 6  root.geometry('300x40') 7 </pre>	<p>команда создаёт корневое (root) окно программы команда меняет заголовок окна команда устанавливает размеры окна</p>
<pre> 8  def button_clicked(): 9      print("Hello World!") 10 11  def close(): 12      root.destroy() 13      root.quit() </pre>	<p>определение функции-обработчика события «нажата кнопка мыши»</p> <p>Функция-обработчик события «заккрытие главного окна». Она останавливает главный цикл приложения и разрушает главное окно. Без неё закрыть программу можно, лишь если завершить процесс интерпретатора Python. Поскольку функция использует глобальную переменную root, объявление самой функции должно следовать после объявления переменной root.</p>
<pre> 15  button = ttk.Button(root, 16      text="Press Me", 17      command=button_clicked) 18  button.pack(fill=BOTH) </pre>	<p>Создание кнопки с текстом «PressMe» и привязка её к функции-обработчику. root можно опускать, т.к. это значение по умолчанию.</p> <p>«Упаковываем» созданную кнопку с помощью менеджера компоновки pack. fill=BOTH (также можно fill="both") указывает кнопке занимать все доступное пространство (по ширине и высоте) на родительском виджете root</p>
<pre> 19 20  root.protocol('WM_DELETE_WINDOW', close) </pre>	<p>Привязываем событие закрытия главного окна с функцией-обработчиком close</p>
<pre> 22  root.mainloop() </pre>	<p>Запускаем главный цикл приложения</p>

## Раздаточный материал № 2 - Основные виджеты (справочно)

### Toplevel/root

Toplevel - окон верхнего уровня. Обычно используется для создания многооконных программ, а также для диалоговых окон.

Методы виджета

title - заголовок окна

overrideredirect - указание оконному менеджеру игнорировать это окно.

Аргументом является True или False. В случае, если аргумент не указан - получаем текущее значение. Если аргумент равен True, то такое окно будет показано оконным

менеджером без обрамления (без заголовка и бордюра). Может быть использовано, например, для создания splashscreen при старте программы.

iconify / deiconify - свернуть / развернуть окно

withdraw - "спрятать" (сделать невидимым) окно. Для того, чтобы снова показать его, надо использовать метод deiconify.

minsize и maxsize - минимальный / максимальный размер окна. Методы принимают два аргумента - ширина и высота окна. Если аргументы не указаны - возвращают текущее значение.

state - получить текущее значение состояния окна. Может возвращать следующие значения: normal (нормальное состояние), icon (показано в виде иконки), iconic (свёрнуто), withdrawn (не показано), zoomed (развёрнуто на полный экран, только для Windows и Mac OS X)

resizable - может ли пользователь изменять размер окна. Принимает два аргумента - возможность изменения размера по горизонтали и по вертикали. Без аргументов возвращает текущее значение.

geometry - устанавливает геометрию окна в формате ширинавысота+х+у (пример: geometry("600x400+40+80") - поместить окно в точку с координатам 40,80 и установить размер в 600x400). Размер или координаты могут быть опущены (geometry("600x400") - только изменить размер, geometry("+40+80") - только переместить окно).

transient - сделать окно зависимым от другого окна, указанного в аргументе. Будет сворачиваться вместе с указанным окном. Без аргументов возвращает текущее значение.

protocol - получает два аргумента: название события и функцию, которая будет вызываться при наступлении указанного события. События могут называться WM\_TAKE\_FOCUS (получение фокуса), WM\_SAVE\_YOURSELF (необходимо сохраниться, в настоящий момент является устаревшим), WM\_DELETE\_WINDOW (удаление окна).

tkraise (синоним lift) и lower - поднимает (размещает поверх всех других окон) или опускает окно. Методы могут принимать один необязательный аргумент: над/под каким окном разместить текущее.

grab\_set - устанавливает фокус на окно, даже при наличии открытых других окон

grab\_release - снимает монопольное владение фокусом ввода с окна

Пример:

```
from Tkinter import *
def window_deleted():
    print u'Окно закрыто'
    root.quit() # явное указание на выход из программы
root = Tk()
root.title(u'Пример приложения')
root.geometry('500x400+300+200') # ширина=500, высота=400, x=300, y=200
root.protocol('WM_DELETE_WINDOW', window_deleted) # обработчик закрытия
окна
root.resizable(True, False) # размер окна может быть изменён только по
горизонтали
root.mainloop()
```

Таким способом можно предотвратить закрытие окна (например, если закрытие окна приведёт к потере введенных пользователем данных).

## Button

Виджет Button - самая обыкновенная кнопка, которая используется в тысячах программ. Примеркода:

```
from Tkinter import *
root = Tk()
button1 = Button(root, text='ok', width=25, height=5, bg='black', fg='red', font='arial 14')
button1.pack()
root.mainloop()
```

За создание, собственно, окна, отвечает класс Tk(), и первым делом нужно создать экземпляр этого класса. Этот экземпляр принято называть root, хотя вы можете назвать его как угодно. Далее создаётся кнопка, при этом мы указываем её свойства (начинать нужно с указания окна, в примере - root). Здесь перечислены некоторые из них:

text - какой текст будет отображён на кнопке (в примере - ок)  
width, height - соответственно, ширина и длина кнопки.  
bg - цвет кнопки (сокращенно от background, в примере цвет - чёрный)  
fg - цвет текста на кнопке (сокращённо от foreground, в примере цвет - красный)  
font - шрифт и его размер (в примере - arial, размер - 14)

## Label

Label - это виджет, предназначенный для отображения какой-либо надписи без возможности редактирования пользователем. Имеет те же свойства, что и перечисленные свойства кнопки.

## Entry

Entry - это виджет, позволяющий пользователю ввести одну строку текста. Имеет дополнительное свойство bd (сокращённо от borderwidth), позволяющее регулировать ширину границы.

borderwidth - ширина бордюра элемента  
bd - сокращение от borderwidth  
width - задаёт длину элемента в знаках.  
show - задает отображаемый символ.

## Text

Text - это виджет, который позволяет пользователю ввести любое количество текста. Имеет дополнительное свойство wrap, отвечающее за перенос (чтобы, например, переносить по словам, нужно использовать значение WORD). Например,

```
from Tkinter import *
root = Tk()
text1 = Text(root, height=7, width=7, font='Arial 14', wrap=WORD)
text1.pack()
root.mainloop()
```

Методы `insert`, `delete` и `get` добавляют, удаляют или извлекают текст. Первый аргумент - место вставки в виде 'x.y', где x – это строка, а y – столбец. Например,

```
text1.insert(1.0,'Добавить Текст\n' в начало первой строки')
text1.delete('1.0', END) # Удалить все
text1.get('1.0', END)    # Извлечь все
```

## Listbox

Listbox - это виджет, который представляет собой список, из элементов которого пользователь может выбирать один или несколько пунктов. Имеет дополнительное свойство `selectmode`, которое, при значении `SINGLE`, позволяет пользователю выбрать только один элемент списка, а при значении `EXTENDED` - любое количество. Пример:

```
from Tkinter import *
root = Tk()
listbox1 = Listbox(root, height=5, width=15, selectmode=EXTENDED)
listbox2 = Listbox(root, height=5, width=15, selectmode=SINGLE)
list1 = [u"Москва", u"Санкт-Петербург", u"Саратов", u"Омск"]
list2 = [u"Канберра", u"Сидней", u"Мельбурн", u"Аделаида"]
for i in list1:
    listbox1.insert(END, i)
for i in list2:
    listbox2.insert(END, i)
listbox1.pack()
listbox2.pack()
root.mainloop()
```

## Frame

Виджет `Frame` (рамка) предназначен для организации виджетов внутри окна. Рассмотрим пример:

```
from tkinter import *
root = Tk()
frame1 = Frame(root, bg='green', bd=5)
frame2 = Frame(root, bg='red', bd=5)
button1 = Button(frame1, text=u'Первая кнопка')
button2 = Button(frame2, text=u'Вторая кнопка')
frame1.pack()
frame2.pack()
button1.pack()
button2.pack()
root.mainloop()
Свойство bd отвечает за толщину края рамки.
```

## Checkbutton

Checkbutton - это виджет, который позволяет отметить „галочкой“ определенный пункт в окне. При использовании нескольких пунктов нужно каждому присвоить свою переменную. Разберем пример:

```

fromtkinter import *
root=Tk()
var1=IntVar()
var2=IntVar()
check1=Checkbutton(root,text=u'1 пункт',variable=var1,onvalue=1,offvalue=0)
check2=Checkbutton(root,text=u'2 пункт',variable=var2,onvalue=1,offvalue=0)
check1.pack()
check2.pack()
root.mainloop()

```

IntVar() - специальный класс библиотеки для работы с целыми числами. variable - свойство, отвечающее за прикрепление к виджету переменной. onvalue, offvalue - свойства, которые присваивают прикрепленной к виджету переменной значение, которое зависит от состояния(onvalue - при выбранном пункте, offvalue - при невыбранном пункте).

## Radiobutton

ВиджетRadiobutton выполняет функцию, схожую с функцией виджетаCheckbutton. Разница в том, что в виджетеRadiobutton пользователь может выбрать лишь один из пунктов. Реализация этого виджета несколько иная, чем виджетаCheckbutton:

```

fromtkinterimport *
root=Tk()
var=IntVar()
rbutton1=Radiobutton(root,text='1',variable=var,value=1)
rbutton2=Radiobutton(root,text='2',variable=var,value=2)
rbutton3=Radiobutton(root,text='3',variable=var,value=3)
rbutton1.pack()
rbutton2.pack()
rbutton3.pack()
root.mainloop()

```

В этом виджете используется уже одна переменная. В зависимости от того, какой пункт выбран, она меняет своё значение. Если присвоить этой переменной какое-либо значение, поменяется и выбранный виджет.

## Scale

Scale (шкала) - это виджет, позволяющий выбрать какое-либо значение из заданного диапазона. Свойства:

orient - как расположена шкала на окне. Возможные значения: HORIZONTAL, VERTICAL (горизонтально, вертикально).

length - длина шкалы.

from\_ - с какого значения начинается шкала.

to - каким значением заканчивается шкала.

tickinterval - интервал, через который отображаются метки шкалы.

resolution - шаг передвижения (минимальная длина, на которую можно передвинуть движок)

Примеркода:

```

fromtkinter import *
root = Tk()
defgetV(root):
    a = scale1.get()
    print "Значение", a
scale1 = Scale(root,orient=HORIZONTAL,length=300,from_=50,to=80,tickinterval=5,
resolution=5)
button1 = Button(root,text=u"Получитьзначение")
scale1.pack()
button1.pack()
button1.bind("<Button-1>",getV)
root.mainloop()

```

Здесь используется специальный метод `get()`, который позволяет снять с виджета определенное значение, и используется не только в `Scale`.

## Scrollbar

Этот виджет даёт возможность пользователю "прокрутить" другой виджет (например, текстовое поле). Необходимо сделать две привязки: `command` полосы прокрутки привязываем к методу `xview/yview` виджета, а `xscrollcommand/yscrollcommand` виджета привязываем к методу `set` полосы прокрутки.

Пример:

```

fromtkinter import *
root = Tk()
text = Text(root, height=3, width=60)
text.pack(side='left')
scrollbar = Scrollbar(root)
scrollbar.pack(side='left')
# перваяпривязка
scrollbar['command'] = text.yview
# втораяпривязка
text['yscrollcommand'] = scrollbar.set
root.mainloop()

```

## Раздаточный материал № 3 - Упаковщики (справочно)

### pack()

Упаковщик `pack()` является самым интеллектуальным (и самым непредсказуемым). При использовании этого упаковщика с помощью свойства `side` нужно указать к какой стороне родительского виджета он должен примыкать. Как правило этот упаковщик используют для размещения виджетов друг за другом (слева направо или сверху вниз). Пример:

```
from tkinter import *
root = Tk()
button1 = Button(text="1")
button2 = Button(text="2")
button3 = Button(text="3")
button4 = Button(text="4")
button5 = Button(text="5")
button1.pack(side='left')
button2.pack(side='top')
button3.pack(side='left')
button4.pack(side='bottom')
button5.pack(side='right')
root.mainloop()
```

Для создания сложной структуры с использованием этого упаковщика обычно используют `Frame`, вложенные друг в друга.

При применении этого упаковщика можно указать следующие аргументы:

`side` ("left"/"right"/"top"/"bottom") - к какой стороне должен примыкать размещаемый виджет.

`fill` (None/"x"/"y"/"both") - необходимо ли расширять пространство предоставляемое виджету.

`expand` (True/False) - необходимо ли расширять сам виджет, чтобы он занял всё предоставляемое ему пространство.

`in_` - явное указание в какой родительский виджет должен быть помещён.

Дополнительные функции

`pack_configure` - синоним для `pack`.

`pack_slaves` (синоним `slaves`) - возвращает список всех дочерних упакованных виджетов.

`pack_info` - возвращает информацию о конфигурации упаковки.

`pack_propagate` (синоним `propagate`) (True/False) - включает/отключает распространение информации о геометрии дочерних виджетов. По умолчанию виджет изменяет свой размер в соответствии с размером своих потомков. Этот метод может отключить такое поведение (`pack_propagate(False)`). Это может быть полезно, если необходимо, чтобы виджет имел фиксированный размер и не изменял его по прихоти потомков.[7]

`pack_forget` (синоним `forget`) - удаляет виджет и всю информацию о его расположении из упаковщика. Позднее этот виджет может быть снова размещён.

## grid()

Этот упаковщик представляет собой таблицу с ячейками, в которые помещаются виджеты.

### Аргументы

row - номер строки, в который помещаем виджет.

rowspan - сколько строк занимает виджет

column - номер столбца, в который помещаем виджет.

columnspan - сколько столбцов занимает виджет.

padx / pady - размер внешней границы (бордюра) по горизонтали и вертикали.

ipadx / ipady - размер внутренней границы (бордюра) по горизонтали и вертикали.

Разница между pad и ipad в том, что при указании pad расширяется свободное пространство, а при ipad расширяется помещаемый виджет.

sticky ("n", "s", "e", "w" или их комбинация) - указывает к какой границе "приклеивать" виджет. Позволяет расширять виджет в указанном направлении. Границы названы в соответствии со сторонами света. "n" (север) - верхняя граница, "s" (юг) - нижняя, "w" (запад) - левая, "e" (восток) - правая.

in\_ - явное указание в какой родительский виджет должен быть помещён.

Для каждого виджета указываем, в какой он находится строке, и в каком столбце. Если нужно, указываем, сколько ячеек он занимает (если, например, нам нужно разместить три виджета под одним, необходимо "растянуть" верхний на три ячейки).  
Пример:

```
entry1.grid(row=0,column=0,columnspan=3)
```

```
button1.grid(row=1,column=0)
```

```
button2.grid(row=1,column=1)
```

```
button3.grid(row=1,column=2)
```

### Дополнительные функции

grid\_configure - синоним для grid.

grid\_slaves (синоним slaves) - см. pack\_slaves.

grid\_info - см. pack\_info.

grid\_propagate (синоним propagate) - см. pack\_propagate.

grid\_forget (синоним forget) - см. pack\_forget.

grid\_remove - удаляет виджет из-под управления упаковщиком, но сохраняет информацию об упаковке. Этот метод удобно использовать для временного удаления виджета.

grid\_bbox (синоним bbox) - возвращает координаты (в пикселях) указанных столбцов и строк.

grid\_location (синоним location) - принимает два аргумента: x и y (в пикселях). Возвращает номер строки и столбца в которые попадают указанные координаты, либо -1 если координаты попали вне виджета.

grid\_size (синоним size) - возвращает размер таблицы в строках и столбцах.

grid\_columnconfigure (синоним columnconfigure) и grid\_rowconfigure (синоним rowconfigure) - важные функции для конфигурирования упаковщика. Методы принимают номер строки/столбца и аргументы конфигурации. Список возможных аргументов:

minsize - минимальная ширина/высота строки/столбца.

weight - "вес" строки/столбца при увеличении размера виджета. 0 означает, что строка/столбец не будет расширяться. Строка/столбец с "весом" равным 2 будет расширяться вдвое быстрее, чем с весом 1.

uniform - объединение строк/столбцов в группы. Строки/столбцы имеющие одинаковый параметр uniform будут расширяться строго в соответствии со своим весом.



pad - размер бордюра. Указывает, сколько пространства будет добавлено к самому большому виджету в строке/столбце.

Пример, текстовый виджет с двумя полосами прокрутки:

```
from tkinter import *
root = Tk()
text = Text(wrap=NONE)
vscrollbar = Scrollbar(orient='vert', command=text.yview)
text['yscrollcommand'] = vscrollbar.set
hscrollbar = Scrollbar(orient='hor', command=text.xview)
text['xscrollcommand'] = hscrollbar.set
# размещаем виджеты
text.grid(row=0, column=0, sticky='nsew')
vscrollbar.grid(row=0, column=1, sticky='ns')
hscrollbar.grid(row=1, column=0, sticky='ew')
# конфигурируем упаковщик, чтобы текстовый виджет расширился
root.rowconfigure(0, weight=1)
root.columnconfigure(0, weight=1)
root.mainloop()
```

## place()

place представляет собой простой упаковщик, позволяющий размещать виджет в фиксированном месте с фиксированным размером. Также он позволяет указывать координаты размещения в относительных единицах для реализации "резинового" размещения. При использовании этого упаковщика, нам необходимо указывать координаты каждого виджета. Например:

```
button1.place(x=0,y=0)
```

Этот упаковщик, хоть и кажется неудобным, предоставляет полную свободу в размещении виджетов на окне.

### Аргументы

anchor ("n", "s", "e", "w", "ne", "nw", "se", "sw" или "center") - какой угол или сторона размещаемого виджета будет указана в аргументах x/y/relx/rely. По умолчанию "nw" - левый верхний

bordermode ("inside", "outside", "ignore") - определяет в какой степени будут учитываться границы при размещении виджета.

in\_ - явное указание в какой родительский виджет должен быть помещён.

x и y - абсолютные координаты (в пикселях) размещения виджета.

width и height - абсолютные ширина и высота виджета.

relx и rely - относительные координаты (от 0.0 до 1.0) размещения виджета.

relwidth и relheight - относительные ширина и высота виджета.

Относительные и абсолютные координаты (а также ширину и высоту) можно комбинировать. Так например, relx=0.5, x=-2 означает размещение виджета в двух пикселях слева от центра родительского виджета, relheight=1.0, height=-2 - высота виджета на два пикселя меньше высоты родительского виджета.

### Дополнительные функции

place\_slaves, place\_forget, place\_info - см. описание аналогичных методов упаковщика pack.

#### Раздаточный материал № 4

<Button-1> – клик левой кнопкой мыши

<Button-2> – клик средней кнопкой мыши

<Button-3> – клик правой кнопкой мыши

<Double-Button-1> – двойной клик левой кнопкой мыши

<Motion> – движение мыши

и т. д.

#### Раздаточный материал № 5

lambda<аргумент(ы)>: <выражение>

#### Раздаточный материал № 6

```
# список целых чисел, которые нужно возвести в квадрат
L = [1, 2, 3, 4]
print(list(map(lambda x: x**2, L)))
```

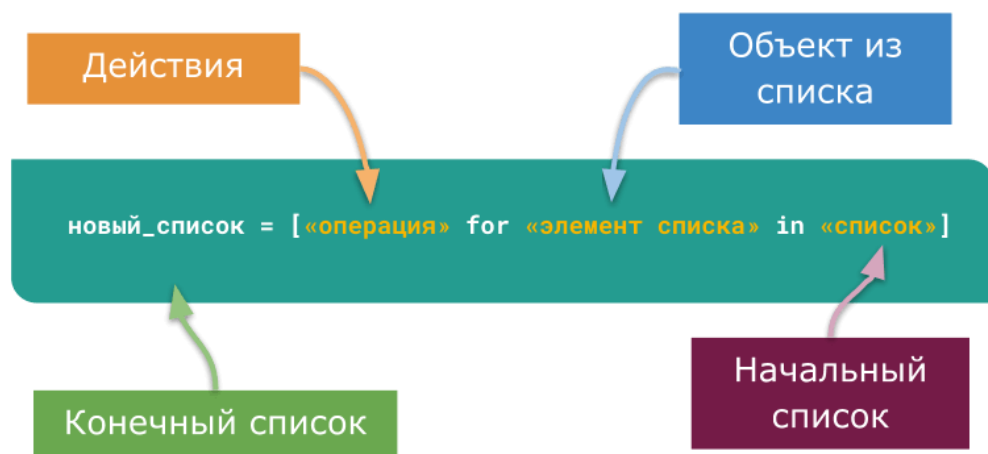
#### Раздаточный материал № 7

```
print(list(filter(lambda x: x % 2 == 0, [1, 3, 2, 5, 20, 21])))
```

#### Раздаточный материал № 8

```
from functools import reduce
print(reduce(lambda x, y: y-x, L)) # работа reduce
# 3 - 1 = 2
# 2 - 2 = 0
# 5 - 0 = 5
# 20 - 5 = 15
# 21 - 15 = 6
```

#### Раздаточный материал № 9



## Раздаточный материал № 10

### #1

```
# винтернет-магазинесегодня 10% скидка на ряд товаров

price = [500, 1200, 800, 600, 150]
price_new = [n * (1 - 0.1) for n in price]

print('Старый прайс', price)
print('Новый прайс', price_new)
```

### #2

```
>>> nums = [n for n in range(1, 6)]
>>> print(nums)
[1, 2, 3, 4, 5]
```

## Раздаточный материал № 11

новый\_список = [«операция» for «элемент списка» in «список» if «условие»]

Важно: здесь невозможно использовать elif, else или другие if

```
price_new1 = [n * (1 - 0.1) for n in price if n < 1000]
print('Новый прайс состоит из стоимости менее 1 тыс. руб.', price_new1)
```

## Раздаточный материал № 12

новый\_список = [«операция» if «условие» for «элемент списка» in «список»]

Важно: условие может дополняться вариантом else (но elif невозможен).

## Раздаточный материал № 13

```
# Дана строка, в которой могут присутствовать буквы любых алфавитов.
# Составить новый список, где напротив каждой буквы будет отмечено,
# является ли она английской или нет

from string import ascii_letters # string импортирован объект ascii_letters,
                                # в котором содержатся только буквы
                                # английского алфавита

letters = 'хытфгтгцзqn' # набор букв из разных алфавитов

# Разграничиваем буквы на английские и не английские
is_eng = [f'{letter}-ДА' if letter in ascii_letters else f'{letter}-НЕТ'
           for letter in letters]
print(is_eng)
```

## Раздаточный материал № 14

```
# Генерация таблицы умножения от 3 до 7
table = [
    [x * y for x in range(3, 8)]
    for y in range(3, 8)]
print(table)
```

## Раздаточный материал № 15

```
# выбрать все гласные буквы из исходной фразы
fraz_a = "я изучаю язык Питон"
new_fraz_a = {i for i in fraza if i in 'аеёиоуэюя'}
print(new_fraz_a)
```

```
# в словаре в качестве значения ключа поместить его квадрат
squares = {i: i * i for i in range(10)}
print(squares)
```

## Раздаточный материал № 16

```
id = [1, 2, 3, 4]
name = ['Меркурий', 'Венера', 'Земля', 'Марс']

rec = zip(id, name) # объединение для двух списков
print(list(rec))

radius = [2439, 6051, 3678, 3376]
rec1 = zip(id, name, radius) # объединение для трех списков
print(list(rec1))

radius_1 = [2439, 6051, 3678]
rec2 = zip(id, name, radius_1) # объединение для трех списков по длине
наименьшего
print(list(rec2))

name_dict_1 = {i: nd for i, nd in zip(id, name)} # создание словаря с
использованием dictcomprehension
print(name_dict_1)

name_dict_2 = dict(zip(id, name)) # создание словаря с использованием
dictcomprehension
print(name_dict_2)

# добавим в словарь новые значения

new_id = [5]
new_name = ['Юпитер']
name_dict_2.update(zip(new_id, new_name))
print(name_dict_2)

# zip и выполнение расчетов
diff = [a-b for a, b in zip(radius, radius[1:])]
print(diff)
```

## Раздаточный материал № 17

```
) >>> lst = [1, 6, 8, 10, 20, 2, 5]
>>> type(lst)
<class 'list'>
>>> lst_it = iter(lst)
>>> type(lst_it)
<class 'list_iterator'>
>>> dir(lst)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__']
>>> dir(lst_it)
['__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__next__', '__str__']
>>> next(lst_it)
1
>>> next(lst_it)
6
>>> next(lst_it)
8
>>> next(lst_it)
10
>>> next(lst_it)
20
>>> next(lst_it)
2
>>> next(lst_it)
5
>>> next(lst_it)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
StopIteration
```

## Раздаточный материал № 18

```
numbers = [6, 57, 4, 7, 68, 95]
```

```
sq = (n**2 for n in numbers)
```

## Раздаточный материал № 19

```
# Вариант 1
def sq_all(numbers):
    for n in numbers:
        yield n ** 2

numbers = [6, 57, 4, 7, 68, 95]
squares = sq_all(numbers)

for i in squares:
    print(i)
# Вариант 2
def sq_all(numbers):
    # оператор for убирается как самостоятельная конструкция
    yield from [n ** 2 for n in numbers]

numbers = [6, 57, 4, 7, 68, 95]
squares = sq_all(numbers)

for i in squares:
    print(i)
```

## Раздаточный материал № 20

```
# В заданной строке найти все прописные буквы, посчитать их количество.
# Использовать библиотеку string

from string import ascii_uppercase

string_new = 'In PyCharm, you can specify third-party standalone
applications and run them as External Tools'
str_1 = [I for I in string_new if I in ascii_uppercase]
print(len(str_1))
print(str_1)
```

## Раздаточный материал № 21 Константы библиотеки string (справочно)

string.ascii\_letters

Объединение констант ascii\_lowercase и ascii\_uppercase описано ниже. Значение не зависит от языкового стандарта.

string.ascii\_lowercase

Строчные буквы 'abcdefghijklmnopqrstuvwxyz'.

string.ascii\_uppercase

Заглавные буквы 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'.

string.digits

Строка '0123456789'.

string.hexdigits

Строка '0123456789abcdefABCDEF'.

string.octdigits

Строка '01234567'.

string.punctuation

Строка символов ASCII, которые считаются символами пунктуации:  
!"#\$%&'()\*+,-./:;<=>?@[\\]^\_`{|}~.

string.printable

Строка символов ASCII, которые считаются печатаемыми. Комбинация digits, ascii\_letters, punctuation и whitespace.

string.whitespace

Строка, содержащая все символы ASCII, считающиеся пробелами. Включает в себя пространство символов, табуляцию, перевод строки, возврат, перевод страницы и вертикальную табуляцию.

## Раздаточный материал № 22

. ^ (читается «карет») \$ \* + ? { } [ ] \ | ( )

### Раздаточный материал № 23

[09] — соответствует числу 0 или 9;  
[0-9] — соответствует любому числу от 0 до 9;  
[абв] — соответствует буквам «а», «б» и «в»;  
[а-г] — соответствует буквам «а», «б», «в» и «г»;  
[а-яё] — соответствует любой букве от «а» до «я»;  
[АБВ] — соответствует буквам «А», «Б» и «В»;  
[А-ЯЁ] — соответствует любой букве от «А» до «Я»;  
[а-яА-ЯёЁ] — соответствует любой русской букве в любом регистре;  
[0-9а-яА-ЯёЁa-zA-z] — любая цифра и любая буква независимо от регистра и языка.

Буква «ё» не входит в диапазон [а-я], а буква «Ё» — в диапазон [А-Я].

[^09] - не цифра 0 или 9;  
[^0-9] - не цифра от 0 до 9;  
[^а-яА-ЯёЁa-zA-Z] - не буква.

### Раздаточный материал № 24

\d — соответствует любой цифре. При указании флага А (ASCII) эквивалентно [0-9];  
\w — соответствует любой букве, цифре или символу подчеркивания. При указании флага А (ASCII) эквивалентно [a-zA-Z0-9\_];  
\s — любой пробельный символ. При указании флага А (ASCII) эквивалентно [\t\n\r\f\v];  
\D — не цифра. При указании флага А (ASCII) эквивалентно [^0-9];  
\W — не буква, не цифра и не символ подчеркивания. При указании флага А (ASCII) эквивалентно [^a-zA-Z0-9\_];  
\S — не пробельный символ. При указании флага А (ASCII) эквивалентно [^\t\n\r\f\v].

В Python 3 поддержка Unicode в регулярных выражениях установлена по умолчанию. При этом все классы трактуются гораздо шире. Так, класс \d соответствует не только десятичным цифрам, но и другим цифрам из кодировки Unicode, — например, дробям, класс \w включает не только латинские буквы, но и любые другие, а класс \s охватывает также неразрывные пробелы. Поэтому на практике лучше явно указывать символы внутри квадратных скобок, а не использовать классы.

### Раздаточный материал № 25

^ Начало текста  
\$ Конец текста  
\b — привязка к началу слова (началом слова считается пробел или любой символ, не являющийся буквой, цифрой или знаком подчеркивания);  
\B — привязка к позиции, не являющейся началом слова.

### Раздаточный материал № 26

<шаблон> = re.compile(<регулярное выражение> [, <флаг>])

## Раздаточный материал № 27

`p = re.compile(r"^\w+$")`  
нужно было бы записать так:  
`p = re.compile("^\\w+$")`

## Раздаточный материал № 28

```
import re # Подключаем модуль
d = "29,12.2009" # Вместо точки указана запятая
p = re.compile(r"^[0-3][0-9] \. [01][0-9] \. [12][09][0-9][0-9]$")
# Символ "\" не указан перед точкой
if p.search(d):
    print("Дата введена правильно")
else:
    print("Дата введена неправильно")
# Так как точка означает любой символ,
# выведет: Дата введена правильно

p = re.compile(r"^[0-3][0-9]\\. [01][0-9]\\. [12][09][0-9][0-9]$")
# Символ "\" указан перед точкой
if p.search(d):
    print("Дата введена правильно")
else:
    print("Дата введена неправильно")
# Так как перед точкой указан символ
# выведет: Дата введена неправильно

p = re.compile(r"^[0-3][0-9] [.] [01][0-9] [.] [12][09][0-9][0-9]$")
# Точка внутри квадратных скобок
if p.search(d):
    print("Дата введена правильно")
else:
    print("Дата введена неправильно")
# Выведет: Дата введена неправильно
```

## Раздаточный материал № 29

```
import re
p = re.compile(r"^.+$") # Точка не соответствует \n
print(p.findall("str1\nstr2\nstr3")) # Ничего не найдено []

p = re.compile(r"^.+$", re.S) # Теперь точка соответствует \n
print(p.findall("str1\nstr2\nstr3")) #
# Строка полностью соответствует ['str1\nstr2\nstr3']

p = re.compile(r"^.+$", re.M) # Многострочный режим
print(p.findall("str1\nstr2\nstr3")) # Получили каждую подстроку ['str1',
# 'str2', 'str3']
```

## Раздаточный материал № 30

```
import re # Подключаем модуль
p = re.compile(r"^[0-9]+$", re.S)
if p.search("245"):
    print("Число") # Выведет: Число
else:
    print("Не число")

if p.search("Строка245"):
    print("Число")
else:
    print("Не число") # Выведет: Не число
```



### Раздаточный материал № 31

```
import re # Подключаем модуль
p = re.compile(r"[0-9]+", re.S)
if p.search("Строка245"):
    print("Число") # Выведет: Число
else:
    print("Не число")
```

### Раздаточный материал № 32

```
# Привязка к началу и концу строки

import re
p = re.compile(r"[0-9]+$", re.S)
if p.search("Строка245"):
    print("Есть число в конце строки")
else:
    print("Нет числа в конце строки")
# Выведет: Есть число в конце строки

p = re.compile(r"^[0-9]+", re.S)
if p.search("Строка245"):
    print("Есть число в начале строки")
else:
    print("Нет числа в начале строки")
# Выведет: Нет числа в начале строки
```

### Раздаточный материал № 33

```
import re

p = re.compile(r"\bpython\b")
print ("Найдено" if p.search ("python") else "Нет")
# выдаст Найдено

print ("Найдено" if p.search("pythonware") else "Нет")
# выдаст Нет

p = re.compile(r"\Bth\B")
print ("Найдено" if p.search("python") else "Нет")
# выдаст Найдено

print ("Найдено" if p.search("this") else "Нет")
# выдаст Нет
```

### Раздаточный материал № 34

```
import re

p = re.compile(r"красн(ая|ое)")
print("Найдено" if p.search("красная") else "Нет")
# выдаст Найдено

print("Найдено" if p.search("красное") else "Нет")
# выдаст Найдено

print("Найдено" if p.search("красный") else "Нет")
# выдаст Нет
```

### Раздаточный материал № 35

{n} — n вхождений символа в строку. Например, шаблон `r"^[0-9]{2}$"` соответствует двум вхождениям любой цифры;

(n,) — n или более вхождений символа в строку. Например, шаблон `r"^[0-9][2, ]$"` соответствует двум и более вхождениям любой цифры;

{n,m} — не менее n и не более m вхождений символа в строку. Числа указываются через запятую без пробела. Например, шаблон `r"^[0-9]{2,4}$"` соответствует от двух до четырех вхождений любой цифры;

\* — ноль или большее число вхождений символа в строку. Эквивалентно комбинации {0,};

+ — одно или большее число вхождений символа в строку. Эквивалентно комбинации {1,};

? — ни одного или одно вхождение символа в строку. Эквивалентно комбинации {0,1}.

### Раздаточный материал № 36

Получим содержимое всех тегов `<b>` вместе с тегами:

```
import re

s = "<b>Text1</b>Text2<b>Text3</b>"
p = re.compile(r"<b>.*</b>", re.S)
print(p.findall(s))

# выдаст ['<b>Text1</b>Text2<b>Text3</b>']
# ожидалось ['<b>Text1</b>', '<b>Text3</b>']
```

### Раздаточный материал № 37

```
p = re.compile(r"<b>.*?</b>", re.S)
print(p.findall(s))

# выдаст ['<b>Text1</b>', '<b>Text3</b>']
```

### Раздаточный материал № 38

```
p = re.compile(r"<b>(.*?)</b>", re.S)
print(p.findall(s))

# выдаст ['Text1', 'Text3']
```

### Раздаточный материал № 39

`match (<Строка>[, <Начальная позиция> [, <Конечная позиция>] ])`

```
import re

p = re.compile(r"[0-9]+")
print("Найдено" if p.match("str123") else "Нет")
# выдаст Нет

print("Найдено" if p.match("str123", 3) else "Нет")
# выдаст Найдено

print("Найдено" if p.match("123str") else "Нет")
# выдаст Найдено
```

#### Раздаточный материал № 40

re.match(<Шаблон>, <Строка>[, <Модификатор>])

```
p = r"[0-9]+"\nprint("Найдено" if re.match(p, "str123") else "Нет")\n# выдаст Нет
```

#### Раздаточный материал № 41

search(<Строка>[, <Начальная позиция>[, <Конечная позиция>]])

```
p = re.compile(r"[0-9]+")\nprint ("Найдено" if p.search("str123") else "Нет")\n# выдаст Найдено\n\nprint ("Найдено" if p.search("123str") else "Нет")\n# выдаст Найдено\n\nprint ("Найдено" if p.search("123str", 3) else "Нет")\n# выдаст Нет
```

#### Раздаточный материал № 42

re. search (<Шаблон>, <Строка>[, <модификатор>])

```
p = r"[0-9]+"\nprint("Найдено" if re.search(p, "str123") else "Нет")\n# выдаст Найдено
```

#### Раздаточный материал № 43

fullmatch (<Строка>[, <Начальная позиция>[, <Конечная позиция>] ])

```
p = re.compile("[Pp]ython")\nprint("Найдено" if p.fullmatch("Python") else "Нет")\n# выдаст Найдено\n\nprint("Найдено" if p.fullmatch("py") else "Нет")\n# выдаст Нет\n\nprint("Найдено" if p.fullmatch("PythonWare") else "Нет")\n# выдаст Нет\n\nprint("Найдено" if p.fullmatch("PythonWare", 0, 6) else "Нет")\n# выдаст Найдено
```

#### Раздаточный материал № 44

re.fullmatch(<Шаблон>, <Строка>[, <Модификатор>])

#### Раздаточный материал № 45

findall(<Строка>[, <Начальная позиция>[, <Конечная позиция>]])

```
import re\n\np = re.compile(r"[0-9]+")\nprint(p.findall ("2007, 2008, 2009, 2010, 2011"))\n# выдаст ['2007', '2008', '2009', '2010', '2011']\n\np = re.compile(r"[a-z]+")\nprint(p.findall("2007, 2008, 2009, 2010, 2011"))\n# выдаст []
```

## Раздаточный материал № 46

```
re.findall(<Шаблон>, <Строка>[, <Модификатор>])
```

## Раздаточный материал № 47

```
finditer(<Строка>[, <Начальная позиция>[, <Конечная позиция>]])
```

## Раздаточный материал № 48 (справочно)

- ♦ I или IGNORECASE — поиск без учета регистра:

```
import re
p = re.compile(r"^[a-яе]+$", re.I | re.U)
print ("Найдено" if p.search("АБВГДЕЕ") else "Нет")
Найдено
p = re.compile(r"^[a-яе]+$", re.U)
print ("Найдено" if p.search("АБВГДЕЕ") else "Нет")
Нет
```

- ♦ M или MULTILINE — поиск в строке, состоящей из нескольких подстрок, разделенных символом новой строки ("\n"). Символ ^ соответствует привязке к началу каждой подстроки, а символ \$ — позиции перед символом перевода строки;

- ♦ S или DOTALL — метасимвол «точка» по умолчанию соответствует любому символу, кроме символа перевода строки (\n). Символу перевода строки метасимвол «точка» будет соответствовать в присутствии дополнительного модификатора. Символ ^ соответствует привязке к началу всей строки, а символ \$ — привязке к концу всей строки:

```
p = re.compile(r"^. $")
print ("Найдено" if p.search("\n") else "Нет")
Нет
```

```
p = re.compile(r"^. $", re.M)
print ("Найдено" if p.search("\n") else "Нет")
Нет
```

```
p = re.compile(r"^. $", re.S)
print ("Найдено" if p.search("\n") else "Нет")
Найдено
```

- ♦ X или VERBOSE — если флаг указан, то пробелы и символы перевода строки будут проигнорированы. Внутри регулярного выражения можно использовать и комментарии:

```
p = re.compile(r""""^ # Привязка к началу строки
[0-9]+ # Строка должна содержать одну цифру (или более)
$      # Привязка к концу строки
""", re.X | re.S)
print("Найдено" if p.search("1234567890") else "Нет")
Найдено
```

```
print("Найдено" if p.search("abcd123") else "Нет")
    Нет
```

♦ А или ASCII — классы `\w`, `\W`, `\b`, `\B`, `\d`, `\D`, `\s` и `\S` будут соответствовать символам в кодировке ASCII (по умолчанию указанные классы соответствуют Unicode-символам);

Флаги `i` и `u` и UNICODE, включающие режим соответствия Unicode-символам классов `\w`, `\W`, `\b`, `\B`, `\d`, `\D`, `\s` и `\S`, сохранены в Python 3 лишь для совместимости с ранними версиями этого языка и никакого влияния на обработку регулярных выражений не оказывают.

♦ L или LOCALE — учитываются настройки текущей локали. Начиная с Python 3.6, могут быть использованы только в том случае, когда регулярное выражение задается в виде значения типов `bytes` или `bytearray`.

### Раздаточный материал № 49

`sub(<Новый фрагмент или ссылка на функцию>, <Строка для замены>  
[, <Максимальное количество замен>])`

```
import re

p = "Это самый сложный урок"
print(re.sub("сложный", "не сложный", p))
# выдаст Это самый не сложный урок
```

### Раздаточный материал № 50

`subn(<Новый фрагмент или ссылка на функцию>, <Строка для замены>  
[, <Максимальное количество замен>])`

```
#Заменим все числа в строке на 0:
p = re.compile(r"[0-9]+")
print(p.subn("0", "2008, 2009, 2010, 2011"))
# выдаст ('0, 0, 0, 0', 4)
```

### Раздаточный материал № 51

`split(<Исходная строка>[, <Лимит>])`

### Раздаточный материал № 52

Содержимое файла `for_split.txt`

Этот файл  
создан для демонстрации;  
работы функции `split`. В результате должен  
получиться; список

```
import re

p = re.compile(r'[\n;,]+' )
with open('for_split.txt', 'r', encoding='utf-8') as file:
    text = file.read()
```

```
reg_name = re.split(p, text)
print(reg_name)

# выдаст ['Этот файл', 'создан для демонстрации', 'работы функции split. В
результате должен', 'получиться', ' список']
```

### Раздаточный материал № 53 (справочно)

#### Однострочные

Нет пустых строк перед или после документации.

Используйте тройные кавычки, даже если документация уместается на одной строке. Потом будет проще её дополнить.

Закрывающие кавычки на той же строке. Это смотрится лучше.

Нет пустых строк перед или после документации.

Однострочная строка документации не должна быть "подписью" параметров функции / метода (которые могут быть получены с помощью интроспекции).

Вставляйте пустую строку до и после всех строк документации (однострочных или многострочных), которые документируют класс - вообще говоря, методы класса разделены друг от друга одной пустой строкой, а строка документации должна быть смещена от первого метода пустой строкой; для симметрии, поставьте пустую строку между заголовком класса и строкой документации. Строки документации функций и методов, как правило, не имеют этого требования.

Строки документации скрипта (самостоятельной программы) должны быть доступны в качестве "сообщения по использованию", напечатанной, когда программа вызывается с некорректными или отсутствующими аргументами (или, возможно, с опцией "-h", для помощи). Такая строка документации должна документировать функции программы и синтаксис командной строки, переменные окружения и файлы. Сообщение по использованию может быть довольно сложным (несколько экранов) и должно быть достаточным для нового пользователя для использования программы должным образом, а также полный справочник со всеми вариантами и аргументами для искушенного пользователя.

Строки документации модуля должны, как правило, перечислять классы, исключения, функции (и любые другие объекты), которые экспортируются модулем, с краткими пояснениями (в одну строчку) каждого из них. (Эти строки, как правило, дают меньше деталей, чем первая строка документации к объекту). Строки документации пакета модулей (т.е. строка документации в `__init__.py`) также должны включать модули и подпакеты.

Строки документации функции или метода должны обобщить его поведение и документировать свои аргументы, возвращаемые значения, побочные эффекты, исключения, дополнительные аргументы, именованные аргументы, и ограничения на вызов функции.

Строки документации класса обобщают его поведение и перечисляют открытые методы и переменные экземпляра. Если класс предназначен для подклассов, и имеет дополнительный интерфейс для подклассов, этот интерфейс должен быть указан отдельно (в строке документации). Конструктор класса должен быть задокументирован в документации метода `__init__`. Отдельные методы должны иметь свои строки документации.

Если класс - подкласс другого класса, и его поведение в основном унаследовано от этого класса, строки документации должны отмечать это и обобщить различия.

## Раздаточный материал № 54

```
def rectangle():
    """Вычисление площади прямоугольника"""
    a = float(input("Ширина %s: " % figure)) # обращение к глобальной
    b = float(input("Высота %s: " % figure)) # переменной figure
    print("Площадь: %.2f" % (a*b))
```

## Раздаточный материал № 55

```
def triangle():
    """Вычисление площади треугольника

    Используется общепринятая формула

    """
    a = float(input("Основание %s: " % figure))
    h = float(input("Высота %s: " % figure))
    print("Площадь: %.2f" % (0.5 * a * h))
```

## Раздаточный материал № 56

```
"""Это описание модуля"""

def rectangle():
    """Вычисление площади прямоугольника"""
    pass

def triangle():
    """Вычисление площади треугольника

    Используется общепринятая формула

    """
    pass

print(rectangle.__doc__)
print(triangle.__doc__)
```

## Раздаточный материал № 57

```
>>> import sys
>>> sys.path
['C:\\Program Files\\JetBrains\\PyCharm Community Edition 2021.1.2\\plugins\\python-
ce\\helpers\\pydev', 'C:\\Program Files\\JetBrains\\PyCharm Community Edition 2021.1.2\\plugins\\python-
ce\\helpers\\third_party\\thriftpy', 'C:\\Program Files\\JetBrains\\PyCharm Community Edition
2021.1.2\\plugins\\python-ce\\helpers\\pydev', 'C:\\Users\\OLGA\\AppData\\Local\\Programs\\Python\\Python38-
32\\python38.zip', 'C:\\Users\\OLGA\\AppData\\Local\\Programs\\Python\\Python38-32\\DLLs',
'C:\\Users\\OLGA\\AppData\\Local\\Programs\\Python\\Python38-32\\lib',
'C:\\Users\\OLGA\\AppData\\Local\\Programs\\Python\\Python38-32',
'C:\\Users\\OLGA\\AppData\\Local\\Programs\\Python\\Python38-32\\lib\\site-packages', 'C:\\PythonProjects\\zab',
'C:/PythonProjects/zab']
```

## Раздаточный материал № 58

```
>>> import main
>>> dir(main)
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__',
'__package__', ...]
```

## Раздаточный материал № 59

```
if __name__ == '__main__':  
    print_hi('PyCharm')
```

## Раздаточный материал № 60

Существует файл mod.py:

```
a = [10, 20, 30]  
print('a =', a)
```

```
>>> import mod  
a = [100, 200, 300]  
>>> import mod  
>>> import mod
```

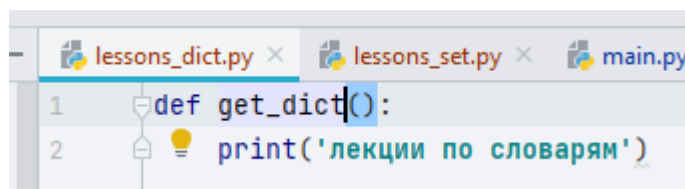
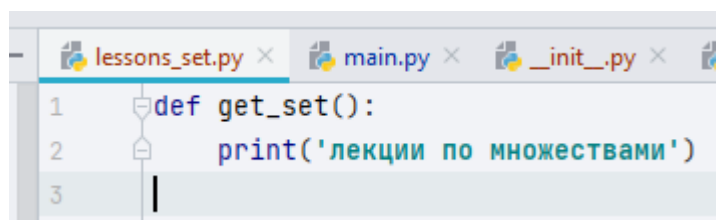
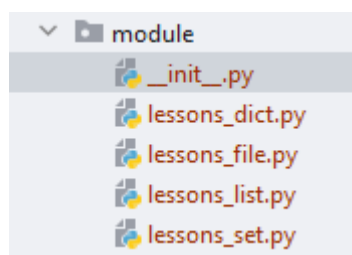
Оператор print() не выполняется при последующем импорте.

## Раздаточный материал № 61

```
>>> import importlib  
>>> importlib.reload(Doc.mod)  
a = [10, 20, 30]  
<module 'Doc.mod' from 'C:\\PythonProjects\\zab\\Doc\\mod.py'>
```

## Раздаточный материал № 62 (выполнить в PyCharm)

В рамках текущего проекта создадим пакет module





```
lessons_file.py x file_doc.py x main.py x modu
1 def get_txt():
2     print('лекции по текстовым файлам')
3
4
5 def get_binary():
6     return "лекции по бинарным файлам"
```

!!! При написании скриптов следить за кодировкой (UTF-8) !!!

В файле main.py выполним импортирование пакета module. Пакеты импортируются также, как и модули.

```
main.py x lessons_file.py x
1 import module
2
3 print(dir(module))
```

Dir() покажет что импортировалось в пространство имен.

```
main x
C:\Users\OLGA\AppData\Local\Programs\Python\Python38-32\python.exe C:/Python
['NAME', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__',
Process finished with exit code 0
```

Видим переменную NAME и служебные переменные, т.е. при импорте пакета автоматически выполнится \_\_init\_\_.py и все что в нем определено импортируется при импорте всего пакета. Но пока без других скриптов пакета. Чтобы импортировать скрипт из модуля необходимо

```
_init_.py x main.py x lessons_file.py x
1 import module.lessons_set
2
3 NAME = "курс лекций по Python"
```

Конструкция module.lessons\_set указывает из какого пакета (module) импортируется нужный модуль (lessons\_set).

После запуска main.py появилось пространство имен lessons\_set, в котором содержатся все объекты скрипта lessons\_set, в частности функция get\_set().

```
__package__, '__path__', '__spec__', 'lessons_set', 'module']
```

Что бы обратиться к функции `get_set()` необходимо

```
1 from .lessons_set import *
2
3 NAME = "курс лекций по Python"
```

Конструкция `.lessons_set` является относительным импортом. Относительный импорт является предпочтительнее, чем абсолютный (`module.lessons_set`), поскольку имя пакета в будущем может меняться и придется изменять название пакета у каждого `import`.

Т.к. при использовании `import *` происходит автоматическое импортирование пространства имен модуля в пространство имен пакета, то в `main.py` теперь достаточно сразу записать `get_set()`, минуя `lessons_set`.

```
1 import module
2
3 module.get_set()
4
5 print(dir(module))
```

Кроме этого удобно использовать и такой вариант `import`, при котором все модули импортируются в программу, но имя пакета не указывается:

```
from . import lessons_set, lessons_dict, lessons_file, lessons_list
```

Символ «\*» не рекомендуется использовать в основных программах (что бы избежать конфликта имен), но для модулей сделано исключение.

Что бы избежать конфликта имен при импортировании (через `import *`) можно контролировать импортируемые данные с помощью переменной `__all__`. Переменная `__all__` должна ссылаться на список, в котором указываются импортируемые объекты.

```
lessons_file.py x file_doc.py x main.py x module\
1  __all__ = ['get_txt', 'get_binary']
2
3  def get_txt():
4      print('лекции по текстовым файлам')
5
6
7  def get_binary():
8      return "лекции по бинарным файлам"
```

В main.py добавим и выполним

```
main.py x lessons_file.py x
1  import module
2
3  module.get_set()
4
5  module.get_txt()
6
7  print(dir(module))
```

```
C:\Users\OLGA\AppData\Local\Pro
лекции по множествами
лекции по текстовым файлам
['NAME', '__builtins__', '__cach
```

## Вложенные пакеты

Внутри пакета module создадим пакет doc, в котором тоже будет свой файл инициализации. В пакете doc создадим еще два файла:

```
module
├── doc
│   ├── __init__.py
│   ├── file_doc.py
│   └── list_doc.py
```

```
file_doc.py x doc\__init__.py x module\__init__.py x mai
1  doc = """Раздаточный материал по файлам"""
2  |
```

В файле `init__.py` пакета doc выполним импорт двух модулей

```
doc\_init_.py x file_doc.py x module\_init_.py
1 from . import list_doc, file_doc
2
```

В файле `init__.py` пакета `module` импортируем пакет `doc`

```
module\_init_.py x doc\_init_.py x file_
1 from .lessons_set import *
2 from .lessons_file import *
3 from .doc import *
4
5 NAME = "курс лекций по Python"
```

А в файле `main.py` обратимся к строке «Раздаточный материал по спискам»

```
main.py x doc\_init_.py x module\_
1 import module
2
3 module.get_set()
4
5 module.get_txt()
6
7 print(module.list_doc.doc)
8
9 print(module.file_doc.doc)
10
11 print(module.get_binary())
12
13 print(dir(module))
14
```

```
main x
C:\Users\OLGA\AppData\Local\Programs\Python\
лекции по множествам
лекции по текстовым файлам
Раздаточный материал по спискам
['NAME', '__builtins__', '__cached__', '__d
```

Из модулей вложенных пакетов можно обращаться к модулям внешнего пакета

```
file_doc.py x main.py x module\_init_.py x doc\_init_.py x lessons_file.  
1 from ..lessons_file import get_binary  
2  
3 doc = """Раздаточный материал по файлам: |""" + get_binary()  
4
```

Конструкция `..lessons_file` означает, что необходимо перейти на уровень выше, т.е. к пакету `module` и от туда взять файл `lessons_file`.

В `main.py` добавим и выполним

```
main.py x doc\_init_.py x module\_init_.py  
1 import module  
2  
3 #module.get_set()  
4  
5 #module.get_txt()  
6  
7 #print(module.list_doc.doc)  
8  
9 print(module.file_doc.doc)  
10  
11 #print(dir(module))  
12
```

```
main x  
C:\Users\OLGA\AppData\Local\Programs\Python\Python38-32\python.exe  
Раздаточный материал по файлам: лекции по бинарным файлам  
  
Process finished with exit code 0
```

Если необходимо обратиться уровнем еще выше, то указываются три точки. Чем выше уровень, тем больше точек необходимо указать.

### Раздаточный материал № 63 (Упражнение № 2)

Создайте пакет `'figures'`, состоящий из трех подпакетов: `'triangle'`, `'circle'`, `'square'`. В каждом подпакете будем иметь файл `code.py`, где создадим ряд функций:

– для пакета `'circle'`: функции `circle_perimeter()` – вычисляет длину окружности, `circle_area()` – вычисляет площадь окружности.

Еще заведем переменную `default_radius = 5`, которая будет скрыта при импорте модуля. Ее назначение – дефолтный радиус для окружности, если пользователь не введет свой.

Обе функции принимают на вход только радиус.

– для пакета 'triangle': функции `triangle_perimeter()` – вычисляет периметр треугольника, `triangle_area()` – вычисляет площадь фигуры.

Дополнительно создадим три переменные (длины сторон треугольника):  $a = 7$ ,  $b = 2$ ,  $c = 8$ , которые также не будут видны при импорте.

На вход функциям передается длина трех сторон (если пользователь ничего не введет, то используются значения по умолчанию).

– для пакета 'square': функции `square_perimeter()` – вычисляет периметр квадрата, `square_area()` – вычисляет площадь фигуры.

Дополнительная переменная  $a = 15$  не доступна при импорте и принимается функциями, если пользователь не предоставил свои размеры стороны квадрата.

Ваша итоговая задача – позволить человеку, загрузившему ваш пакет, иметь возможность напрямую импортировать все функции из подпакетов.

Например, он может написать так: `from figures import circle_area`.

Также вы, как разработчик, после написания всей библиотеки решили поменять ее имя на 'figures'.

Постарайтесь сделать код таким, чтобы это не заставило вас переписывать все внутренние импорты с учетом нового именования.

#### Раздаточный материал № 64



#### Раздаточный материал № 65

users				
id	sex	name	level	score
1	2	Сергей	2	26116
2	2	Михаил	2	13832
3	1	Мария	1	0

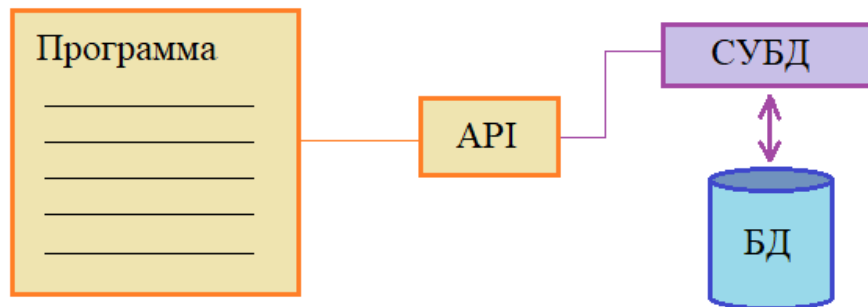
games			
id	user_id	score	time
15	1	1514	1559470775
16	1	2310	1559471049
17	1	1706	1559471132
18	1	11	1559568764
19	1	1260	1559568871
20	1	20	1559569080
21	1	1165	1559627469
22	1	1259	1559627528

## Раздаточный материал № 66

games			
id	user_id	score	time
15	1	1514	1559470775
16	1	2310	1559471049
17	1	1706	1559471132

users				
id	sex	name	level	score
1	2	Сергей	2	26116
2	2	Михаил	2	13832
3	1	Мария	1	0

## Раздаточный материал № 67



## Раздаточный материал № 68

execute(SQL)

SQL (Structured Query Language)

DBeaver - <https://dbeaver.io/>

## Раздаточный материал № 69

```
import sqlite3 as sq

with sq.connect('saper.db') as con:
    cur = con.cursor()
    #cur.execute("DROP TABLE IF EXISTS users")
    cur.execute("""CREATE TABLE IF NOT EXISTS users (
        user_id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT NOT NULL,
        sex INTEGER NOT NULL DEFAULT 1,
        old INTEGER,
        score INTEGER
    ) """)
```

## Раздаточный материал № 70

\*.db, \*.db3, \*.sqlite и \*.sqlite3

## Раздаточный материал № 71

NULL – значение NULL;  
INTEGER – целочисленный тип (занимает от 1 до 8 байт);  
REAL – вещественный тип (8 байт в формате IEEE);  
TEXT – строковый тип (в кодировке данных базы, обычно UTF-8);  
BLOB (двоичные данные, хранятся «как есть», например, для небольших изображений).

## Раздаточный материал № 72

INSERT INTO <table\_name> (<column\_name1>, <column\_name2>, ...) VALUES (<value1>, <value2>, ...)

или так:

INSERT INTO <table\_name> VALUES (<value1>, <value2>, ...)

## Раздаточный материал № 73

INSERT INTO users VALUES (1, 'Михаил', 1, 19, 1000)

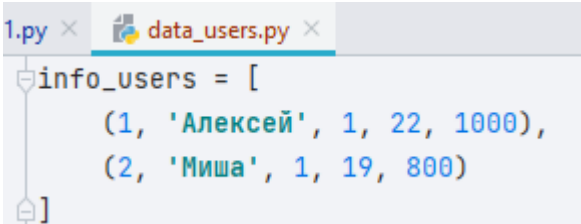
## Раздаточный материал № 74

```
cur.execute("INSERT INTO users VALUES (1, 'Алексей', 1, 22, 1000)")
```

## Раздаточный материал № 75

```
cur.executemany("INSERT INTO users VALUES (?, ?, ?, ?, ?)", info_users)
```

Содержимое info\_users



```
1.py x data_users.py x
info_users = [
    (1, 'Алексей', 1, 22, 1000),
    (2, 'Миша', 1, 19, 800)
]
```

## Раздаточный материал № 76

SELECT col1, col2, ... FROM <table\_name>

Например:

SELECT name, old, score FROM users

На выходе получим выборку из трех столбцов и набора всех записей из таблицы users.



### **Раздаточный материал № 77**

SELECT col1, col2, ... FROM <table\_name> WHERE <условие>

Например, отберем все записи со значением очков меньше 1000:

SELECT \* FROM users WHERE score < 1000

### **Раздаточный материал № 78**

= или ==, >, <, >=, <=, !=, BETWEEN

Например:

SELECT \* FROM users WHERE score BETWEEN 500 AND 1000

Будут выбраны все записи с числом очков в диапазоне от 500 до 1000. Или, так:

### **Раздаточный материал № 79**

AND – условное И: exp1 AND exp2. Истинно, если одновременно истинны exp1 и exp2.

OR – условное ИЛИ: exp1 OR exp2. Истинно, если истинно exp1 или exp2 или оба выражения.

NOT – условное НЕ: NOT exp. Преобразует ложное условие в истинное и, наоборот, истинное – в ложное.

IN – вхождение во множество значений: col IN (val1, val2, ...)

NOT IN – не вхождение во множество значений: col NOT IN (val1, val2, ...)

Самый высокий приоритет имеет операция NOT. Приоритет у операции AND выше, чем у OR.

Например:

SELECT \* FROM users WHERE old > 20 AND score < 1000

Выбирает игроков возрастом более 20 лет и с числом очков менее 1000. Или, так:

SELECT \* FROM users WHERE old IN (19, 32) AND score < 1000

Создается выборка из игроков возрастом 19 или 32 года и числом очков менее 1000. Следующий запрос:

SELECT \* FROM users WHERE old IN (19, 32) AND score > 300 OR sex = 1

SELECT \* FROM users WHERE (old IN (19, 32) OR sex = 1) AND score > 300

### **Раздаточный материал № 80**

SELECT \* FROM users WHERE score < 1000 ORDER BY old

## Раздаточный материал № 81

SELECT \* FROM users WHERE score < 1000 ORDER BY old DESC

## Раздаточный материал № 82

```
with sq.connect('saper.db') as con:
    cur = con.cursor()

    cur.execute("SELECT * FROM users WHERE score > 1000 ORDER BY score DESC")
    result = cur.fetchall()
    print(result)

#результат: [(4, 'Мария', 2, 18, 1500), (5, 'Александр', 1, 20, 1100)]
```

## Раздаточный материал № 83

```
cur.execute("SELECT * FROM users WHERE score > 1000 ORDER BY score DESC")
for result in cur:
    print(result)
```

## Раздаточный материал № 84

fetchmany(size) – возвращает число записей не более size;  
fetchone() – возвращает первую запись.

Например

```
cur.execute("SELECT * FROM users")
result1 = cur.fetchone()
result2 = cur.fetchmany(2)
print(result1)
print(result2)

#результат (1, 'Алексей', 1, 22, 1000)
#[ (2, 'Миша', 1, 19, 800), (3, 'Сергей', 1, 19, 900) ]
```

## Раздаточный материал № 85

UPDATE имя\_таблицы SET имя\_столбца = новое\_значение WHERE условие

Например,

можно менять значения сразу несколько столбцов записи, перечисляя их через запятую:

UPDATE users SET score = 700, old = 45 WHERE old > 40

всем игрокам женского пола увеличить число очков на 500:

UPDATE users SET score = score+500 WHERE sex = 2

обратиться к игрокам по имени и указать им определенное число очков:

UPDATE users SET score = 1500 WHERE name LIKE 'Федор'

### Раздаточный материал № 86

% - любое продолжение строки;

\_ - любой символ;

Например:

```
UPDATE users SET score = score+100 WHERE name LIKE 'M%'
```

### Раздаточный материал № 87

```
cur.execute("UPDATE users SET score = score+100 WHERE name LIKE 'M%'")
```

### Раздаточный материал № 88

DELETE FROM имя\_таблицы WHERE условие

Например:

```
DELETE FROM users WHERE user_id = 5
```

### Раздаточный материал № 89

```
cur.execute("DELETE FROM users WHERE user_id = 5")
```

### Раздаточный материал № 90

```
class <Название класса> [( <Класс1>[, ..., <КлассN>] )]:  
    [ """"Строка документирования """" ]  
    <Описание атрибутов и методов>
```

### Раздаточный материал № 91

<Экземпляр класса> = <Название класса> ([<Параметры>])

При обращении к методам класса:

### Раздаточный материал № 92

<Экземпляр класса>.<Имя метода> ([<Параметры>])

### Раздаточный материал № 93

<Экземпляр класса>.<Имя атрибута>