北京大学 | 暑期学校 Summer School

Python语言基础与应用06

北京大学 陈斌

2019.07.09

目录

- 面向对象案例分析
- 例外处理
- 生成器和推导式
- import和名字空间
- 时间相关模块/算术模块
- 文本文件读写
- 数据库sqlite3
- 简单的图形界面easygui

面向对象案例分析：空战游戏

- 基于pygame的空战游戏
- 用户通过鼠标控制战斗机（Fighter）左右移动，自动发射炮弹（Shell）
- 战斗机要躲避3种不同生命值的障碍物（Obstacle）
- 如果炮弹与障碍物碰撞，则炮弹消失，障碍物减生命值
- 如果战斗机与障碍物碰撞，则战斗机坠毁，游戏结束。

https://github.com/chbpku/bdfz.courses/tree/master/python.programming/samples/airstrike

战斗机类的定义FighterClass

```python
# 创建一个pygame Sprite类的子类：战斗机
class FighterClass(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.image.load("fighter.png")
        self.rect = self.image.get_rect()
        self.rect.center = [320, 540]  # 飞机的初始位置，320代表在初始位置在窗口中心

    def fire(self, type):  # 发射炮弹
        global shells
        shell = ShellClass(
            type + ".png", [self.rect.centerx, self.rect.centery - 20], type)
        shells.add(shell)

    def fly(self, x, y):  # 飞机在左右、上下方向移动，并且保证都不出界
        self.rect.centerx = x
        if self.rect.centerx < 20:
            self.rect.centerx = 20
        if self.rect.centerx > 620:
            self.rect.centerx = 620
        self.rect.centery = y
        if self.rect.centery < 20:
            self.rect.centery = 20
        if self.rect.centery > 620:
            self.rect.centery = 620
```

障碍物类的定义ObstacleClass

```python
# 创建一个pygame Sprite类的子类：障碍物，包含三种
class ObstacleClass(pygame.sprite.Sprite):
    def __init__(self, image_file, location, type):
        pygame.sprite.Sprite.__init__(self)
        self.image_file = image_file
        self.image = pygame.image.load(image_file)
        self.rect = self.image.get_rect()
        self.rect.center = location
        self.type = type
        if type == "p1":   # 障碍物P1的生命值是1000
            self.life = 1000
        elif type == "p2":
            self.life = 2000
        elif type == "p3":
            self.life = 100000000
        self.passed = False

    def update(self):   # 障碍物自上而下移动
        global speed
        self.rect.centery += speed[1]
        if self.rect.centery > 672 or self.life <= 0:   # 如果障碍物已移出屏幕
            self.kill()  # 删掉障碍物
```

各种属性

炮弹类的定义ShellClass

```python
class ShellClass(ObstacleClass):
    def __init__(self, image_file, location, type):
        ObstacleClass.__init__(self, image_file, location, type)
        self.direction = 0
        if type == "shell1":
            self.direction = 1

    def update(self):   # 炮弹自下而上移动
        global speed
        self.rect.centerx += self.direction * speed[1] * 0.8
        self.rect.centery -= speed[1]
        if self.rect.centery < -32 or self.rect.centerx < - \
                32 or self.rect.centerx > 672:   # 如果炮弹已移出屏幕
            self.kill()   # 删掉炮弹
```

对象生成和交互

```python
# 主循环
while running:
    c = c + 1
    clock.tick(60)  # 循环每秒运行60次
    maxs = 6 + (points / 600)
    if maxs < 6:
        maxs = 6
    for event in pygame.event.get():  # 检查按键事件
        if event.type == pygame.QUIT:  # 如果关闭游戏窗口，就同时退出程序
            running = False
        elif event.type == pygame.MOUSEMOTION:  # 鼠标移动
            x = event.pos[0]
            y = event.pos[1]
            fighter.fly(x, y)
        elif event.type == pygame.KEYDOWN:  # 按下键盘上的q键时，退出游戏
            if event.key == pygame.K_q:
                running = False
    if c == 8:  # 发射炮弹的频率
        fighter.fire("shell")
        c = 0
```

对象之间的交互

```python
hit = pygame.sprite.groupcollide(
    shells,
    obstacles,
    dokilla=False,
    dokillb=False)  # 碰撞检测，计算得分
if hit:
    for shell in hit:
        for enemy in hit[shell]:
            enemy.life -= 1000
            if enemy.type == "p1":
                points = points + 10
                bomb.play()
            elif enemy.type == "p2":
                points = points + 20
                bomb.play()
            elif enemy.type == "p3":
                bomb.play()
        if shell.type == "shell":
            shell.kill()
```

例外处理Exception

```
>>> lst=[1,2,3]
>>> for i in range(4):
        print(lst[i])


1
2
3
Traceback (most recent call last):
  File "<pyshell#178>", line 2, in <module>
    print(lst[i])
IndexError: list index out of range
>>>
```

- 代码运行可能会意外各种错误：
  - 语法错误：Syntax Error
  - 除以0错误：ZeroDivisionError
  - 列表下标越界：IndexError
  - 类型错误：TypeError…
- 事先无法预料，如：
  - 由用户输入/交互引起
  - 由外部数据引起
  - 由设备连接等引起

```
>>> c=int(input("Please input number:"))
Please input number:ABCD
Traceback (most recent call last):
  File "<pyshell#167>", line 1, in <module>
    c=int(input("Please input number:"))
ValueError: invalid literal for int() with base 10: 'ABCD'
>>>
```

例外处理Exception Handling

- 错误会引起程序中止退出
- 如果希望掌控意外，就需要在可能出错误的地方设置陷阱捕捉错误
  - try: # 为缩进的代码设置陷阱
  - except: # 处理错误的代码
  - else: # 没有出错执行的代码
  - finally: # 无论出错否，都执行的代码

```python
try:
    print('try...')
    r = 10 / 'xyz'
    print('result:', r)
except TypeError as e:
    print('TypeError:', e)
except ZeroDivisionError as e:
    print('ZeroDivisionError:', e)
else:
    print('no error!')
finally:
    print('finally...')
print('END')
```

```
try...
TypeError: unsupported operand type(s) for /: 'int' and 'str'
finally...
END
```

推导式

- 可以用来生成列表、字典和集合的语句
  - [<表达式> for <变量> in <可迭代对象> if <逻辑条件>]
  - {<键值表达式>:<元素表达式> for <变量> in <可迭代对象> if <逻辑条件>}
  - {<元素表达式> for <变量> in <可迭代对象> if <逻辑条件>}

```
>>> [x*x for x in range(10)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>>
>>> {'K%d'%(x,):x**3 for x in range(10)}
{'K2': 8, 'K8': 512, 'K5': 125, 'K6': 216, 'K3': 27, 'K9': 729, 'K0': 0,
'K7': 343, 'K1': 1, 'K4': 64}
>>>
>>> {x*x for x in range(10)}
{0, 1, 64, 4, 36, 9, 16, 49, 81, 25}
>>>
>>> {x+y for x in range(10) for y in range(x)}
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17}
```

推导式

```
>>> [x+y for x in range(10) for y in range(x)]
[1, 2, 3, 3, 4, 5, 4, 5, 6, 7, 5, 6, 7, 8, 9, 6, 7, 8, 9, 10, 11, 7, 8, 9
, 10, 11, 12, 13, 8, 9, 10, 11, 12, 13, 14, 15, 9, 10, 11, 12, 13, 14, 15
, 16, 17]
>>>
>>> [x*x for x in range(10) if x % 2 == 0]
[0, 4, 16, 36, 64]
>>>
>>> [x.upper() for x in [1, 'abc', 'xyz', True] if isinstance(x, str)]
['ABC', 'XYZ']
```

生成器推导式

- 与推导式一样语法：
  - (<元素表达式> for <变量> in <可迭代对象> if <逻辑条件>)
- 返回一个生成器对象，也是可迭代对象
- 但生成器并不立即产生全部元素，仅在要用到元素的时候才生成，可以极大节省内存

```
>>> agen = (x*x for x in range(10))
>>> agen
<generator object <genexpr> at 0x1078f5620>
>>> for n in agen:
        print (n)


0
1
4
9
16
25
36
```

生成器函数

- 如果生成器较复杂，一行表达式无法容纳，可以定义生成器函数
- 生成器函数的定义与普通函数相同，只是将return换成了yield
  - yield会立即返回一个值
  - 但在下一次迭代生成器函数的时候，会从yield语句后的语句继续执行，直到再次yield返回，或终止
  - return语句则不同，它会终止函数的执行，下次调用函数会重新执行函数

```
def even_number(max):
    n = 0
    while n < max:
        yield n
        n += 2

for i in even_number(10):
    print (i)
```

```
============== RESTART:
0
2
4
6
8
>>>
```

上机练习

- 编写程序，输入两个数，输出它们的商，采用例外处理来处理两种错误，给出用户友好的提示信息
  - 1）除数为0
  - 2）输入了非数值
- 编写一个推导式，生成包含100以内所有勾股数(i,j,k)的列表
- 编写一个生成器函数，能够生成斐波那契数列

```
def fib():
    ....
for fn in fib():
    print (fn)
    if fn>1000:
        break
```

Python引用扩展模块

- import <模块> [as <别名>]
  - 将模块中的函数等名称导入当前程序
  - “命名空间”namespace
  - 引用方法：<模块>.<名称>
- dir(<名称>)函数
  - 列出名称的属性
- help(<名称>)函数
  - 显示参考手册
- from <模块> import <名称>
  - 导入模块的部分名称

```
>>> import time
>>> dir(time)
['_STRUCT_TM_ITEMS', '__doc__', '__loader__', '__name__', '__package__', '__spec__', 'altzone', 'asctime', 'clock', 'ctime', 'daylight', 'get_clock_info', 'gmtime', 'localtime', 'mktime', 'monotonic', 'perf_counter', 'process_time', 'sleep', 'strftime', 'strptime', 'struct_time', 'time', 'timezone', 'tzname', 'tzset']
>>> time.tzname
('CST', 'CST')
>>> help(time.time)
Help on built-in function time in module time:

time(...)
    time() -> floating point number

    Return the current time in seconds since the Epoch.
    Fractions of a second may be present if the system
clock provides them.

>>> print(time.time())
1490280256.450634
```

时间相关：calendar模块

- 跟日历相关的若干函数和类，可以生成文本形式的日历
- calendar.calendar(<年>)
- calendar.month(<年>,<月>)
  - 返回多行字符串
- calendar.isleap(<年>)
  - 判别闰年
- calendar.prmonth(<年>,<月>)
- calendar.prcal(<年>)

```
>>> import calendar
>>> calendar.month(2019, 7)
'     July 2019\nMo Tu We Th Fr Sa Su\n 1
2  3  4  5  6  7\n 8  9 10 11 12 13 14\n15
16 17 18 19 20 21\n22 23 24 25 26 27 28\n29
30 31\n'
>>> print(calendar.month(2019, 7))
     July 2019
Mo Tu We Th Fr Sa Su
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31

>>> calendar.isleap(2019)
False
```

时间相关：datetime模块

- 有4个主要的类
  - date处理年月日
  - time处理时分秒、毫秒
  - datetime处理日期加时间
  - timedelta处理时段（时间间隔）
- 常用函数／方法
  - datetime.date.today()
  - datetime.datetime.now()
  - datetime.datetime.isoformat()
- 两个时间相减就是timedelta

```
>>> import datetime
>>> datetime.date.today()
datetime.date(2019, 7, 8)
>>> d = datetime.datetime.now()
>>> d
datetime.datetime(2019, 7, 8, 23, 20, 54, 963571)
>>> d.isoformat()
'2019-07-08T23:20:54.963571'
>>> d - datetime.timedelta(days = 2)
datetime.datetime(2019, 7, 6, 23, 20, 54, 963571)
```

2天前

```
>>> d2 = datetime.datetime.now()
>>> d2 - d
datetime.timedelta(seconds=98, microseconds=854541)
>>> d2
datetime.datetime(2019, 7, 8, 23, 22, 33, 818112)
>>> dt = d2 - d
>>> type(dt)
<class 'datetime.timedelta'>
```

时间相关：time模块

- time.time()浮点数表示的现在时间
  - 从1970-1-1 0:0:0开始的秒数
- time.struct_time结构化时间类
  - time.localtime(<纪元时间>)->结构
  - time.gmtime(<纪元时间>)->结构
  - time.mktime(<结构化时间>)->纪元时间
- time.strftime(<格式>)表示格式化输出（结构化）时间
- time.strptime(<字串>,<格式>)按照格式识别字串，返回时间

```
>>> import time
>>> n = time.time()
>>> n
1562599454.688421
>>> time.localtime(n)
time.struct_time(tm_year=2019, tm_mon=7, tm_mday=8,
tm_hour=23, tm_min=24, tm_sec=14, tm_wday=0, tm_yday
=189, tm_isdst=0)
>>> time.gmtime(n)
time.struct_time(tm_year=2019, tm_mon=7, tm_mday=8,
tm_hour=15, tm_min=24, tm_sec=14, tm_wday=0, tm_yday
=189, tm_isdst=0)
>>> p = time.localtime(n)
>>> time.mktime(p)
1562599454.0
>>> p = time.gmtime(n)
>>> time.mktime(p)
1562570654.0
>>> time.strftime("%Y%m%d %H%M%S", p)
'20190708 152414'
>>> time.strptime("20190708", "%Y%m%d")
time.struct_time(tm_year=2019, tm_mon=7, tm_mday=8,
tm_hour=0, tm_min=0, tm_sec=0, tm_wday=0, tm_yday=18
9, tm_isdst=-1)
```

基本模块简介：算术

- math：常用的算术函数、三角函数、幂指数等等
- cmath：支持复数的math函数
- decimal：十进制定点数
  - 十进制小数
  - 不再有浮点数的误差
- fractions：有理数，比例
  - 进行分数运算

```
>>> import decimal as d
>>> 0.1+0.2
0.30000000000000004
>>> d.Decimal("0.1")+d.Decimal("0.2")
Decimal('0.3')
>>> import fractions as f
>>> f.Fraction("2/3")
Fraction(2, 3)
>>> f.Fraction("2/3")+f.Fraction("3/4")
Fraction(17, 12)
>>>
```

```
>>> from decimal import Decimal as D
>>> D("3.14159265351234567323211234")+D("7.29231456343423423534 3")
```

基本模块简介：算术

- random：随机数
  - random.randint(a,b)
  - random.randrange(start,stop,step)
  - random.choice(seq)
  - random.sample(seq, n)
- statistics：一些统计函数
  - 平均值：mean
  - 中位数：median
  - 标准偏差：stdev/pstdev

```
>>> import random
>>> random.choice(["apple", "pie", "tea"])
'tea'
>>> random.randrange(10)
4
>>> random.randint(1,10)
5
```

```
>>> random.sample(["apple", "pie", "tea", "milk"], 2)
['tea', 'milk']
```

```
>>> import statistics
>>> statistics.mean([12,34,56])
34.0
>>> statistics.median([12,34,56,100])
45.0
>>> statistics.median([12,34,56,100,101])
56
```

持久化：shelve

- 将任何数据对象，保存到文件中去
- 类似字典形式访问，可读可写
  - `import shelve`
  - `f = shelve.open(<文件名>)`
  - `f[key] = value`
  - `value = f[key]`
  - `del f[key]`
  - `f.close()`

无作用

```
import shelve

d = shelve.open(filename)   # open -- fi
                            # library

d[key] = data               # store data
                            # using an e
data = d[key]               # retrieve a
                            # if no such
del d[key]                  # delete dat
                            # if no such

flag = key in d             # true if th
klist = list(d.keys())      # a list of

# as d was opened WITHOUT writeback=Tru
d['xx'] = [0, 1, 2]         # this works
d['xx'].append(3)           # *this does

# having opened d without writeback=Tru
temp = d['xx']              # extracts t
temp.append(5)              # mutates th
d['xx'] = temp              # stores the

# or, d=shelve.open(filename,writeback=
# d['xx'].append(5) and have it work as
# consume more memory and make the d.cl

d.close()                   # close it
```

文本文件读写：内置文件对象

- 内置的文本文件处理函数
  - f = open(<文件名>, <模式>)
  - f.readline()：返回一行
    - 如果返回None说明到文件尾
  - f.readlines()：返回所有行，列表
  - f.writelines(<字符串列表>)：写入文本行
  - f.close()
  - with语句可以自动调用close

```
>>> f=open("my.txt", "w")
>>> f.writelines(["apple\n", "pie\n"])
>>> f.close()
>>> f=open("my.txt", "r")
>>> f.readlines()
['apple\n', 'pie\n']
>>> f.close()
```

f2.py - /Us

```
with open("1.txt", "r") as f:
    for a in f:
        print(a)
```

上机练习

- 给算法计时，看看递归生成斐波那契数列的每个数（n=1~30）各需要多长时间?
  - 可以导出到excel画成一个折线图看看
- 将一篇文章写入一个文本文件。
  - 从yahoo.com网站摘一段英文新闻，保存为文本文件（txt）；
  - 读出文本文件，统计单词数输出。
  - 读出文本文件，随机输出其中的10个单词。

数据库操纵模块：sqlite3

- 无服务器端的微型关系数据库
  - 广泛应用在小型系统/移动app
- SQLite数据库访问接口
  - 连接：connect(dbname)
  - 执行SQL：execute(SQL,参数)
  - executemany(SQL,参数迭代)
  - executescript(多条SQL)
  - 游标对象cursor迭代获得数据记录
  - 关闭：close()

数据库操纵模块：sqlite3

```
>>> import sqlite3
>>> conn = sqlite3.connect("test.db")
>>> conn.execute("create table person (id text, name text, age integer)")
<sqlite3.Cursor object at 0x1163088f0>
>>> conn.execute("insert into person values (?,?,?)", ("A01", "John", 23))
<sqlite3.Cursor object at 0x1163089d0>
>>> plst=[("B01", "Tom", 18), ("B02", "Mike", 20), ("C01", "Alice", 22)]
>>> conn.executemany("insert into person values (?,?,?)", plst)
<sqlite3.Cursor object at 0x1163088f0>
>>> conn.commit()
```

插入多条记录

数据库操纵模块：sqlite3

```
>>> cur = conn.execute("select id, name, age from person where name=?", ("Tom",))
>>> for r in cur:
        print (r)


('B01', 'Tom', 18)
>>> cur = conn.execute("delete from person where id=?", ("A01",))
>>> cur = conn.execute("select * from person")
>>> for r in cur:
        print (r)


('B01', 'Tom', 18)
('B02', 'Mike', 20)
('C01', 'Alice', 22)
>>> conn.close()
>>> 
```

数据库操纵模块：sqlite3

- 查看数据库中有哪些表

```
>>> conn = sqlite3.connect("test.db")
>>> cur = conn.execute("select * from sqlite_master")
>>> for r in cur:
        print (r)


('table', 'person', 'person', 2, 'CREATE TABLE person
```

- 查看表的结构

```
>>> cur = conn.execute("pragma table_info(person)")
>>> for r in cur:
        print (r)


(0, 'id', 'text', 0, None, 0)
(1, 'name', 'text', 0, None, 0)
(2, 'age', 'integer', 0, None, 0)
>>>
```

数据库操纵模块：sqlite3

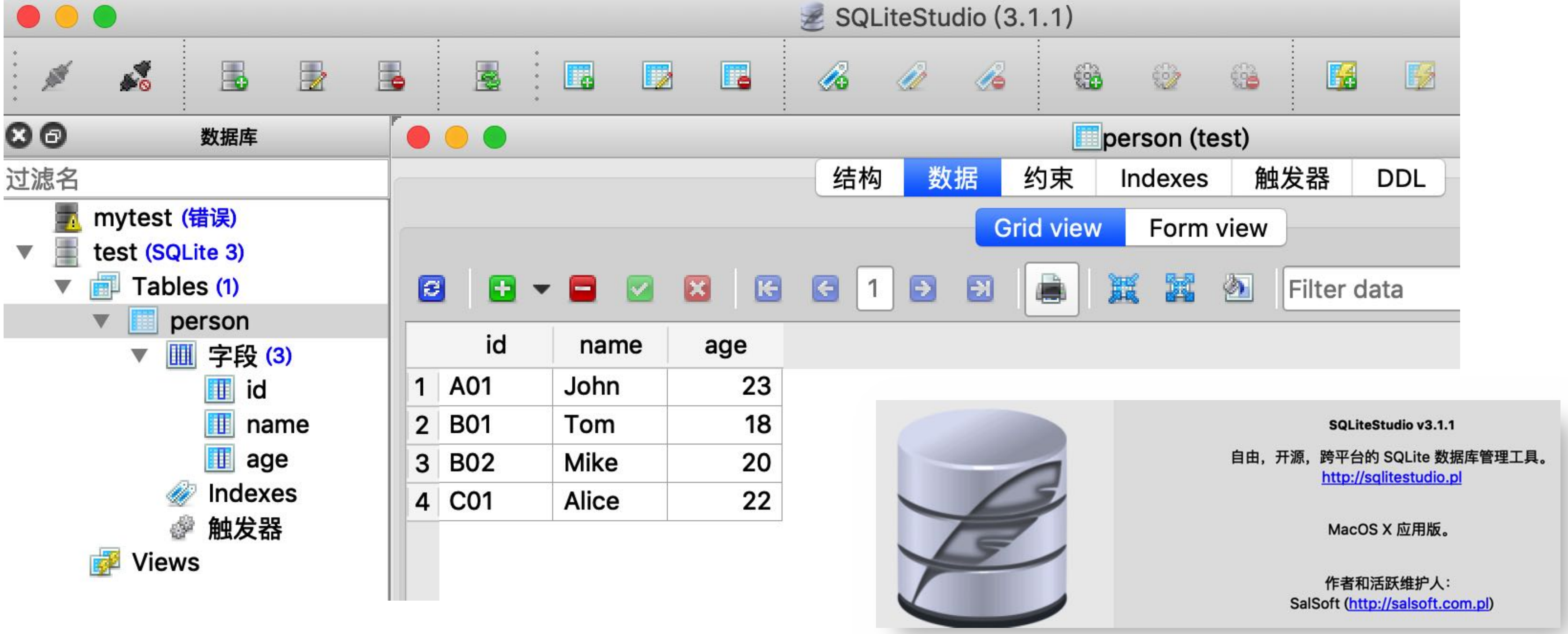
- 数据类型的对应关系
- SQLite支持类型
  - NULL, TEXT, INTEGER
  - REAL, BLOB
- Python对应的类型
  - None, str, int
  - float, bytes
- 很多SQLite数据库图形化工具
  - SQLiteStudio

- 执行SQL语句脚本
  - executescript(多条SQL)

```
import sqlite3

con = sqlite3.connect(":memory:")
cur = con.cursor()
cur.executescript("""
    create table person(
        firstname,
        lastname,
        age
    );

    create table book(
        title,
        author,
        published
    );

    insert into book(title, author, published)
    values (
        'Dirk Gently''s Holistic Detective Agency',
        'Douglas Adams',
        1987
    );
    """)
```

SQLiteStudio

https://sqlitestudio.pl



图形用户界面：easygui

- 可以显示各种对话框、文本框、选择框与用户交互
    - easygui.egdemo()演示
    - easygui.msgbox
    - easygui.fileopenbox
    - easygui.choicebox
    - easygui.textbox
    - easygui.passwordbox
- 可以做出简单的图形界面程序

```python
import easygui as g
import sys
while 1:
    g.msgbox('嗨，欢迎进入第一个GUI制作的小游戏~')
    msg = '你希望学习到什么知识呢？'
    title = '互动小游戏'
    choices = ['琴棋书画', '四书五经', '程序编写', '逆向分析']
    choice = g.choicebox(msg, title, choices)
    # note that we convert the choice to string, in case the user
    # cancelled the choice, and we got None.
    g.msgbox('你的选择是：' + str(choice), '结果')
    msg = '你希望重新开始小游戏么？'
    title = '请选择'
    if g.ccbox(msg, title):     # Show a Continue/Cancel dialog
        pass                    # user choose Continue
    else:
        sys.exit(0)             # user choose Cancel
```

上机练习

- 利用sqlite3模块，结合easygui
- 先下载sqlitestudio，创建一个数据库menu，包含一个表drink
  - 字段：名称name/类别category/价格price
- 创建一个GUI程序，有2个功能
  - 功能1：输入一个饮料的信息，插入数据库
  - 功能2：输入饮料名称，查询饮料信息