

## 前言

告别枯燥，60秒学会一个小例子！

目前已发布[Python之路.pdf V1.0](#)，包括：[Python之基](#)，[Python之正](#)，[Python之例](#)，[Python之能](#)章节，共计143个小例子。欢迎关注《Python小例子》官方公众号：



## 一、Python之基

[Python之基](#)主要总结Python常用内置函数及用法，它们在Python中被最高频的使用，所以务必掌握。[v1.0](#)一共包括59个。

### 1 求绝对值

绝对值或复数的模

```
In [1]: abs(-6)
Out[1]: 6
```

### 2 元素都为真

接受一个迭代器，如果迭代器的所有元素都为真，那么返回True，否则返回False

```
In [2]: all([1,0,3,6])
Out[2]: False

In [3]: all([1,2,3])
Out[3]: True
```

### 3 元素至少一个为真

接受一个迭代器，如果迭代器里至少有一个元素为真，那么返回 `True`，否则返回 `False`

```
In [4]: any([0,0,0,[]])
Out[4]: False

In [5]: any([0,0,1])
Out[5]: True
```

### 4 ascii展示对象

调用对象的`repr()`方法，获得该方法的返回值

```
In [1]: class Student():
...:     def __init__(self,id,name):
...:         self.id = id
...:         self.name = name
...:     def __repr__(self):
...:         return 'id = '+self.id +', name = '+self.name

In [2]: print(xiaoming)
id = 001, name = xiaoming

In [3]: ascii(xiaoming)
Out[3]: 'id = 001, name = xiaoming'
```

### 5 十转二

将十进制转换为二进制

```
In [1]: bin(10)
Out[1]: '0b1010'
```

### 6 十转八

将十进制转换为八进制

```
In [1]: oct(9)
Out[1]: '0o11'
```

## 7 十转十六

将十进制转换为十六进制

```
In [1]: hex(15)
Out[1]: '0xf'
```

## 8 判断是真是假

测试一个对象是True, 还是False.

```
In [1]: bool([0,0,0])
Out[1]: True

In [2]: bool([])
Out[2]: False

In [3]: bool([1,0,1])
Out[3]: True
```

## 9 字符串转字节

将一个字符串转换成字节类型

```
In [1]: s = "apple"

In [2]: bytes(s,encoding='utf-8')
Out[2]: b'apple'
```

## 10 转为字符串

将字符类型、数值类型等转换为字符串类型

```
In [1]: i = 100

In [2]: str(i)
Out[2]: '100'
```

## 11 是否可调用

判断对象是否可被调用, 能被调用的对象就是一个 `callable` 对象, 比如函数 `str`, `int` 等都是可被调用的, 但是例子4中 `xiaoming` 实例是不可被调用的:

```
In [1]: callable(str)
Out[1]: True

In [2]: callable(int)
Out[2]: True

In [3]: xiaoming
Out[3]: id = 001, name = xiaoming

In [4]: callable(xiaoming)
Out[4]: False
```

## 12 十转ASCII

查看十进制整数对应的ASCII字符

```
In [1]: chr(65)
Out[1]: 'A'
```

## 13 ASCII转十

查看某个ASCII字符对应的十进制数

```
In [1]: ord('A')
Out[1]: 65
```

## 14 静态方法

`classmethod` 装饰器对应的函数不需要实例化，不需要 `self` 参数，但第一个参数需要是表示自身类的 `cls` 参数，可以用来调用类的属性，类的方法，实例化对象等。

```
In [1]: class Student():
...:     def __init__(self, id, name):
...:         self.id = id
...:         self.name = name
...:     def __repr__(self):
...:         return 'id = '+self.id +', name = '+self.name
...:     @classmethod
...:     def f(cls):
...:         print(cls)
```

## 15 执行字符串表示的代码

将字符串编译成python能识别或可执行的代码，也可以将文字读成字符串再编译。

```
In [1]: s = "print('helloworld')"  
  
In [2]: r = compile(s,"<string>", "exec")  
  
In [3]: r  
Out[3]: <code object <module> at 0x0000000005DE75D0, file  
<string>, line 1>  
  
In [4]: exec(r)  
helloworld
```

## 16 创建复数

创建一个复数

```
In [1]: complex(1,2)  
Out[1]: (1+2j)
```

## 17 动态删除属性

删除对象的属性

```
In [1]: delattr(xiaoming, 'id')  
  
In [2]: hasattr(xiaoming, 'id')  
Out[2]: False
```

## 18 转为字典

创建数据字典

```
In [1]: dict()  
Out[1]: {}  
  
In [2]: dict(a='a',b='b')  
Out[2]: {'a': 'a', 'b': 'b'}  
  
In [3]: dict(zip(['a','b'],[1,2]))  
Out[3]: {'a': 1, 'b': 2}  
  
In [4]: dict([('a',1),('b',2)])  
Out[4]: {'a': 1, 'b': 2}
```

## 19 一键查看对象所有方法

不带参数时返回当前范围内的变量、方法和定义的类型列表；带参数时返回参数的属性，方法列表。

```
In [96]: dir(xiaoming)
Out[96]:
['__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattr__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',

 'name']
```

## 20 取商和余数

分别取商和余数

```
In [1]: divmod(10,3)
Out[1]: (3, 1)
```

## 21 枚举对象

返回一个可以枚举的对象，该对象的`next()`方法将返回一个元组。

```
In [98]: s = ["a","b","c"]
...: for i ,v in enumerate(s,1):
...:     print(i,v)
...:
1 a
2 b
3 c
```

## 22 计算表达式

将字符串`str`当成有效的表达式来求值并返回计算结果取出字符串中内容

```
In [99]: s = "1 + 3 +5"
...: eval(s)
...:
Out[99]: 9
```

## 23 执行`compile`

执行字符串或`compile`方法编译过的字符串，没有返回值

```
In [74]: s = "print('helloworld')"

In [75]: r = compile(s,"<string>", "exec")

In [76]: r
Out[76]: <code object <module> at 0x0000000005DE75D0, file
"<string>", line 1>

In [77]: exec(r)
helloworld
```

## 24 过滤器

过滤器，构造一个序列，等价于

```
[ item for item in iterables if function(item)]
```

在函数中设定过滤条件，逐一循环迭代器中的元素，将返回值为`True`时的元素留下，形成一个`filter`类型数据。

```
In [101]: fil = filter(lambda x: x>10,[1,11,2,45,7,6,13])

In [102]: list(fil)
Out[102]: [11, 45, 13]
```

## 25 转为浮点类型

将一个字符串或整数转换为浮点数

```
In [103]: float(3)
Out[103]: 3.0
```

## 26 字符串格式化

格式化输出字符串，`format(value, format_spec)`实质上是调用了`value`的`format(format_spec)`方法。

```
In [104]: print("i am {0},age{1}".format("tom",18))
i am tom,age18
```

3.1415926	{:.2F}	3.14	保留小数点后两位
3.1415926	{:+.2f}	+3.14	带符号保留小数点后两位
-1	{:+.2f}	-1.00	带符号保留小数点后两位
2.71828	{:.0f}	3	不带小数
5	{:0>2d}	05	数字补零 (填充左边, 宽度为2)
5	{:x<4d}	5xxx	数字补x (填充右边, 宽度为4)
10	{:x<4d}	10xx	数字补x (填充右边, 宽度为4)
1000000	{:,}	1,000,000	以逗号分隔的数字格式
0.25	{:.2%}	25.00%	百分比格式
1000000000	{:.2e}	1.00e+09	指数记法
18	{:>10d}	' 18'	右对齐 (默认, 宽度为10)
18	{:<10d}	'18 '	左对齐 (宽度为10)
18	{:^10d}	' 18 '	中间对齐 (宽度为10)

## 27 冻结集合

创建一个不可修改的集合。

```
In [105]: frozenset([1,1,3,2,3])
Out[105]: frozenset({1, 2, 3})
```

## 28 动态获取对象属性

获取对象的属性

```
In [106]: getattr(xiaoming,'name')
Out[106]: 'xiaoming'
```

## 29 对象是否有这个属性

```
In [110]: hasattr(xiaoming,'name')
Out[110]: True

In [111]: hasattr(xiaoming,'id')
Out[111]: False
```

## 30 返回对象的哈希值



返回对象的哈希值

```
In [112]: hash(xiaoming)
Out[112]: 6139638
```

### 31 一键帮助

返回对象的帮助文档

```
In [113]: help(xiaoming)
Help on Student in module __main__ object:

class Student(builtins.object)
|   Methods defined here:
|
|   __init__(self, id, name)
|
|   __repr__(self)
|
|   -----
|
|   Data descriptors defined here:
|
|   __dict__
|       dictionary for instance variables (if defined)
|
|   __weakref__
|       list of weak references to the object (if defined)
```

### 32 对象门牌号

返回对象的内存地址

```
In [115]: id(xiaoming)
Out[115]: 98234208
```

### 33 获取用户输入

获取用户输入内容

```
In [116]: input()
aa
Out[116]: 'aa'
```

### 34 转为整型

`int(x, base=10)`, `x`可能为字符串或数值, 将`x`转换为一个普通整数。如果参数是字符串, 那么它可能包含符号和小数点。如果超出了普通整数的表示范围, 一个长整数被返回。

```
In [120]: int('12',16)
Out[120]: 18
```

### 35 实例对应类型

判断`object`是否为类`classinfo`的实例, 是返回`true`

```
In [20]: class Student():
...:     def __init__(self,id,name):
...:         self.id = id
...:         self.name = name
...:     def __repr__(self):
...:         return 'id = '+self.id +', name
= '+self.name
...:

In [21]: xiaoming = Student('001','xiaoming')

In [22]: isinstance(xiaoming,Student)
Out[22]: True
```

### 36 父子关系鉴定

如果`class`是`classinfo`类的子类, 返回`True`:

```
In [27]: class undergraduate(Student):
...:     def studyClass(self):
...:         pass
...:     def attendActivity(self):
...:         pass
...:

In [28]: issubclass(undergraduate,Student)
Out[28]: True

In [29]: issubclass(object,Student)
Out[29]: False

In [30]: issubclass(Student,object)
Out[30]: True
```

如果`class`是`classinfo`元组中某个元素的子类, 也会返回`True`

```
In [26]: issubclass(int,(int,float))
Out[26]: True
```

## 37 创建迭代器类型

返回一个可迭代对象, `sentinel`可省略

```
In [72]: lst = [1,3,5]

In [73]: for i in iter(lst):
...:     print(i)
...:
1
3
5
```

`sentinel` 理解为迭代对象的哨兵, 一旦迭代到此元素, 立即终止:

```
In [81]: class TestIter(object):
...:     def __init__(self):
...:         self.l=[1,3,2,3,4,5]
...:         self.i=iter(self.l)
...:     def __call__(self): #定义了__call__方法的
类的实例是可调用的
...:         item = next(self.i)
...:         print ("__call__ is called,which
would return",item)
...:         return item
...:     def __iter__(self): #支持迭代协议(即定义有
__iter__()函数)
...:         print ("__iter__ is called!!")
...:         return iter(self.l)
...:

In [82]: t = TestIter()
...: t1 = iter(t, 3)
...: for i in t1:
...:     print(i)
...:
__call__ is called,which would return 1
1
__call__ is called,which would return 3
```

## 38 求序列元素长度

返回对象的长度 (元素个数)

```
In [83]: dic = {'a':1,'b':3}

In [84]: len(dic)
Out[84]: 2
```

## 39 转列表类型

返回可变序列类型

```
In [85]: list(map(lambda x: x%2==1, [1,3,2,4,1]))
Out[85]: [True, True, False, False, True]
```

## 40 f映射到元素上

返回一个将 *function* 应用于 *iterable* 中每一项并输出其结果的迭代器:

```
In [85]: list(map(lambda x: x%2==1, [1,3,2,4,1]))
Out[85]: [True, True, False, False, True]
```

可以传入多个 *iterable* 对象, 输出长度等于最短序列的长度:

```
In [88]: list(map(lambda x,y: x%2==1 and y%2==0,
[1,3,2,4,1],[3,2,1,2]))
Out[88]: [False, True, False, False]
```

## 41 可迭代对象最大值

返回最大值:

```
In [99]: max(3,1,4,2,1)
Out[99]: 4

In [100]: max((),default=0)
Out[100]: 0

In [89]: di = {'a':3,'b1':1,'c':4}
In [90]: max(di)
Out[90]: 'c'

In [102]: a =
[{'name':'xiaoming','age':18,'gender':'male'},{'name':'
...: xiaohong','age':20,'gender':'female'}]
In [104]: max(a,key=lambda x: x['age'])
Out[104]: {'name': 'xiaohong', 'age': 20, 'gender':
'female'}
```

## 42 可迭代对象最小值

返回最小值, 参考求可迭代对象最大值函数 `max`

## 43 下一个元素

返回可迭代对象的下一个元素

```
In [129]: it = iter([5,3,4,1])

In [130]: next(it)
Out[130]: 5

In [131]: next(it)
Out[131]: 3

In [132]: next(it)
Out[132]: 4

In [133]: next(it)
Out[133]: 1

In [134]: next(it,0) #迭代到头，默认返回值为0
Out[134]: 0

In [135]: next(it)
-----
-----
StopIteration                                Traceback (most
recent call last)
<ipython-input-135-bc1ab118995a> in <module>
----> 1 next(it)

StopIteration:
```

## 44 所有对象之根

返回一个没有特征的新对象。object 是所有类的基类。

```
In [137]: o = object()

In [138]: type(o)
Out[138]: object
```

## 45 打开文件

返回文件对象

```
In [146]: fo = open('D:/a.txt',mode='r', encoding='utf-8')

In [147]: fo.read()
Out[147]: '\uffefflife is not so long,\nI use Python to
play.'
```

mode取值表:

字符	意义
'r'	读取（默认）
'w'	写入，并先截断文件
'x'	排它性创建，如果文件已存在则失败
'a'	写入，如果文件存在则在末尾追加
'b'	二进制模式
't'	文本模式（默认）
'+'	打开用于更新（读取与写入）

## 46 次幂

base为底的exp次幂，如果mod给出，取余

```
In [149]: pow(3, 2, 4)
Out[149]: 1
```

## 47 打印

```
In [5]: lst = [1,3,5]

In [6]: print(lst)
[1, 3, 5]

In [7]: print(f'lst: {lst}')
lst: [1, 3, 5]

In [8]: print('lst:{}'.format(lst))
lst:[1, 3, 5]

In [9]: print('lst:',lst)
lst: [1, 3, 5]
```

## 48 创建属性的两种方式

返回 property 属性，典型的用法：

```
class C:
    def __init__(self):
        self._x = None

    def getx(self):
        return self._x

    def setx(self, value):
```

```

        self._x = value

    def delx(self):
        del self._x
    # 使用property类创建 property 属性
    x = property(getx, setx, delx, "I'm the 'x'
property.")

```

使用python装饰器，实现与上完全一样的效果代码：

```

class C:
    def __init__(self):
        self._x = None

    @property
    def x(self):
        return self._x

    @x.setter
    def x(self, value):
        self._x = value

    @x.deleter
    def x(self):
        del self._x

```

## 49 创建range序列

- 1) range(stop)
- 2) range(start, stop[,step])

生成一个不可变序列：

```

In [153]: range(11)
Out[153]: range(0, 11)

In [154]: range(0,11,1)
Out[154]: range(0, 11)

```

## 50 反向迭代器

返回一个反向的 iterator:

```
In [155]: rev = reversed([1,4,2,3,1])

In [156]: for i in rev:
...:     print(i)
...:
1
3
2
4
1
```

## 51 四舍五入

四舍五入，ndigits代表小数点后保留几位：

```
In [11]: round(10.022222, 3)
Out[11]: 10.022

In [12]: round(10.05,1)
Out[12]: 10.1
```

## 52 转为集合类型

返回一个set对象，可实现去重：

```
In [159]: a = [1,4,2,3,1]

In [160]: set(a)
Out[160]: {1, 2, 3, 4}
```

## 53 转为切片对象

*class slice(start, stop[, step])*

返回一个表示由 range(start, stop, step) 所指定索引集的 slice对象，它让代码可读性、可维护性大大增强。

```
In [13]: a = [1,4,2,3,1]

In [14]: my_slice_meaning = slice(0,5,2)

In [15]: a[my_slice_meaning]
Out[15]: [1, 2, 1]
```

## 54 拿来就用的排序函数

排序：



```

In [174]: a = [1,4,2,3,1]

In [175]: sorted(a,reverse=True)
Out[175]: [4, 3, 2, 1, 1]

In [178]: a =
[{'name': 'xiaoming', 'age': 18, 'gender': 'male'}, {'name': '
...: xiaohong', 'age': 20, 'gender': 'female'}]
In [180]: sorted(a, key=lambda x: x['age'], reverse=False)
Out[180]:
[{'name': 'xiaoming', 'age': 18, 'gender': 'male'},
 {'name': 'xiaohong', 'age': 20, 'gender': 'female'}]

```

## 55 求和函数

求和:

```

In [181]: a = [1,4,2,3,1]

In [182]: sum(a)
Out[182]: 11

In [185]: sum(a,10) #求和的初始值为10
Out[185]: 21

```

## 56 转元组

`tuple()` 将对象转为一个不可变的序列类型，元组。

```

In [16]: i_am_list = [1,3,5]
In [17]: i_am_tuple = tuple(i_am_list)
In [18]: i_am_tuple
Out[18]: (1, 3, 5)

```

## 57 查看对象类型

`class type(name, bases, dict)`

传入一个参数时，返回 *object* 的类型:

```

In [186]: type(xiaoming)
Out[186]: __main__.Student

In [187]: type(tuple())
Out[187]: tuple

```

## 58 聚合迭代器

创建一个聚合了来自每个可迭代对象中的元素的迭代器：

```
In [188]: x = [3,2,1]
In [189]: y = [4,5,6]
In [190]: list(zip(y,x))
Out[190]: [(4, 3), (5, 2), (6, 1)]

In [191]: a = range(5)
In [192]: b = list('abcde')
In [193]: b
Out[193]: ['a', 'b', 'c', 'd', 'e']
In [194]: [str(y) + str(x) for x,y in zip(a,b)]
Out[194]: ['a0', 'b1', 'c2', 'd3', 'e4']
```

## 59 查看变量所占字节数

```
In [1]: import sys

In [2]: a = {'a':1,'b':2.0}

In [3]: sys.getsizeof(a) # 占用240个字节
Out[3]: 240
```

## 二、Python之正则

**Python之正则**主要总结通过20个例子，入门Python正则表达式。之所以将正则列为一章，是因为字符串处理无所不在，正则毫无疑问是最简洁和高效的处理方法。后面的**Python之例**，**Python之能**章节也会多次使用正则表达式做一些字符串处理相关的工作。

```
import re
```

### 1 查找第一个匹配串

```
s = 'i love python very much'
pat = 'python'
r = re.search(pat,s)
print(r.span()) #(7,13)
```

### 2 查找所有1

```
s = '山东省潍坊市青州第1中学高三1班'
pat = '1'
r = re.finditer(pat,s)
for i in r:
    print(i)

# <re.Match object; span=(9, 10), match='1'>
# <re.Match object; span=(14, 15), match='1'>
```

### 3 \d 匹配数字[0-9]

```
s = '一共20行代码运行时间13.59s'
pat = r'\d+' # +表示匹配数字(\d表示数字的通用字符)1次或多次
r = re.findall(pat,s)
print(r)
# ['20', '13', '59']
```

我们想保留13.59而不是分开，请看4

### 4 ?表示前一个字符匹配0或1次

```
s = '一共20行代码运行时间13.59s'
pat = r'\d+\.\?\d+' # ?表示匹配小数点(\.)0次或1次
r = re.findall(pat,s)
print(r)
# ['20', '13.59']
```

### 5 ^ 匹配字符串的开头

```
s = 'This module provides regular expression matching
operations similar to those found in Perl'
pat = r'^[emrt]' # 查找以
r = re.findall(pat,s)
print(r)
# [],因为字符串的开头是字符`T`, 不在emrt匹配范围内, 所以返回为空
```

### 6 re.I 忽略大小写

```
s = 'This module provides regular expression matching
operations similar to those found in Perl'
pat = r'^[emrt]' # 查找以
r = re.compile(pat,re.I).search(s)
print(r)
# <re.Match object; span=(0, 1), match='T'> 表明字符串的开头
在匹配列表中
```

## 7 使用正则提取单词

这是不准确版本，请参看第9个

```
s = 'This module provides regular expression matching
operations similar to those found in Perl'
pat = r'\s[a-zA-Z]+'
r = re.findall(pat,s)
print(r) #[' module', ' provides', ' regular', '
expression', ' matching', ' operations', ' similar', '
to', ' those', ' found', ' in', ' Perl']
```

## 8 只捕获单词，去掉空格

使用 `()` 捕获，这是不准确版本，请参看第9个

```
s = 'This module provides regular expression matching
operations similar to those found in Perl'
pat = r'\s([a-zA-Z]+)'
r = re.findall(pat,s)
print(r) #['module', 'provides', 'regular', 'expression',
'matching', 'operations', 'similar', 'to', 'those',
'found', 'in', 'Perl']
```

## 9 补充上第一个单词

上面第8，看到提取单词中未包括第一个单词，使用 `?` 表示前面字符出现0次或1次，但是此字符还有表示贪心或非贪心匹配含义，使用时要谨慎。

```
s = 'This module provides regular expression matching
operations similar to those found in Perl'
pat = r'\s?([a-zA-Z]+)'
r = re.findall(pat,s)
print(r) #['This', 'module', 'provides', 'regular',
'expression', 'matching', 'operations', 'similar', 'to',
'those', 'found', 'in', 'Perl']
```

## 10 使用split函数直接分割单词

使用以上方法分割单词，不是简洁的，仅仅为了演示。分割单词最简单还是使用 `split` 函数。

```
s = 'This module provides regular expression matching
operations similar to those found in Perl'
pat = r'\s+'
r = re.split(pat,s)
print(r) # ['This', 'module', 'provides', 'regular',
'expression', 'matching', 'operations', 'similar', 'to',
'those', 'found', 'in', 'Perl']
```

## 11 提取以m或t开头的单词，忽略大小写

下面出现的结果不是我们想要的，原因出在 `?` 上！

```
s = 'This module provides regular expression matching
operations similar to those found in Perl'
pat = r'\s?([mt][a-zA-Z]*)' # 查找以
r = re.findall(pat,s)
print(r) # ['module', 'matching', 'tions', 'milar', 'to',
'those']
```

## 12 使用^查找字符串开头的单词

综合11和12得到所有以m或t开头的单词

```
s = 'This module provides regular expression matching
operations similar to those found in Perl'
pat = r'^([mt][a-zA-Z]*)\s' # 查找以
r = re.compile(pat,re.I).findall(s)
print(r) # ['This']
```

## 13 先分割，再查找满足要求的单词

使用 `match` 表示是否匹配

```
s = 'This module provides regular expression matching
operations similar to those found in Perl'
pat = r'\s+'
r = re.split(pat,s)
res = [i for i in r if re.match(r'[mMtT]',i)]
print(res) # ['This', 'module', 'matching', 'to', 'those']
```

## 14 贪心匹配

尽可能多的匹配字符

```
content='<h>dadedadsad</h>
<div>graph</div>bb<div>math</div>cc'
pat=re.compile(r"<div>(.*?)</div>") #贪婪模式
m=pat.findall(content)
print(m) # ['graph</div>bb<div>math']
```

## 15 非贪心匹配

与14相比，仅仅多了一个问号(`?`)，得到结果完全不同。

```

content='<h>ddedadsad</h>
<div>graph</div>bb<div>math</div>cc'
pat=re.compile(r"<div>(.*?)</div>") #贪婪模式
m=pat.findall(content)
print(m) # ['graph', 'math']

```

与14比较可知，贪心匹配和非贪心匹配的区别，后者是字符串匹配后立即返回，见好就收。

## 16 含有多种分割符

使用 `split` 函数

```

content = 'graph math,,english;chemistry' # 这种
pat=re.compile(r"[\s\,\;\,;]+" ) #贪婪模式
m=pat.split(content)
print(m) # ['graph', 'math', 'english', 'chemistry']

```

## 17 替换匹配的子串

`sub` 函数实现对匹配子串的替换

```

content="hello 12345, hello 456321"
pat=re.compile(r'\d+') #要替换的部分
m=pat.sub("666",content)
print(m) # hello 666, hello 666

```

## 18 爬取百度首页标题

```

import re
from urllib import request

#爬虫爬取百度首页内容
data=request.urlopen("http://www.baidu.com/").read().decode()

#分析网页,确定正则表达式
pat=r'<title>(.*?)</title>'

result=re.search(pat,data)
print(result) <re.Match object; span=(1358, 1382),
match='<title>百度一下, 你就知道</title>'>

result.group() # 百度一下, 你就知道

```

## 19 常用元字符总结

```
. 匹配任意字符
^ 匹配字符串始位置
$ 匹配字符串中结束的位置
* 前面的原子重复0次1次多次
? 前面的原子重复一次或者0次
+ 前面的原子重复一次或多次
{n} 前面的原子出现了 n 次
{n,} 前面的原子至少出现 n 次
{n,m} 前面的原子出现次数介于 n-m 之间
( ) 分组,需要输出的部分
```

## 20 常用通用字符总结

```
\s 匹配空白字符
\w 匹配任意字母/数字/下划线
\W 和小写 w 相反, 匹配任意字母/数字/下划线以外的字符
\d 匹配十进制数字
\D 匹配除了十进制数以外的值
[0-9] 匹配一个0-9之间的数字
[a-z] 匹配小写英文字母
[A-Z] 匹配大写英文字母
```

以上就是Python中正则模块的基本使用总结，里面有循序渐进的优化分析过程，这些虽然是中间过程，但是对于正则小白而言，了解这些很有必要。笔者对于正则的理解和使用也比较肤浅，如有总结不到位之处，恳请指正。

## 三、Python之例

**Python之例** 章中每个例子大都10行左右，1.0版本一共包括 36 个小例子，都是很有意思的小例子。

### 1 链式比较

```
i = 3
print(1 < i < 3) # False
print(1 < i <= 3) # True
```

### 2 不用else和if实现计算器

```
from operator import *

def calculator(a, b, k):
    return {
```

```

        '+': add,
        '-': sub,
        '*': mul,
        '/': truediv,
        '**': pow
    }[k](a, b)

calculator(1, 2, '+') # 3
calculator(3, 4, '**') # 81

```

### 3 链式操作

```

from operator import (add, sub)

def add_or_sub(a, b, oper):
    return (add if oper == '+' else sub)(a, b)

add_or_sub(1, 2, '-') # -1

```

### 4 求字符串的字节长度

```

def str_byte_len(mystr):
    return (len(mystr.encode('utf-8')))

str_byte_len('i love python') # 13(个字节)
str_byte_len('字符') # 6(个字节)

```

### 5 寻找第n次出现位置

```

def search_n(s, c, n):
    size = 0
    for i, x in enumerate(s):
        if x == c:
            size += 1
        if size == n:
            return i
    return -1

print(search_n("fdasadfadf", "a", 3))# 结果为7, 正确
print(search_n("fdasadfadf", "a", 30))# 结果为-1, 正确

```



## 6 去掉最高分、最低分求平均

```
#方法1:
def score_mean(lst):
    lst.sort()
    lst_2=lst[1:(len(lst)-1)]
    return round((sum(lst_2)/len(lst_2)),1)

lst=[9.1, 9.0,8.1, 9.7, 19,8.2, 8.6,9.8]
score_mean(lst)

#方法2:
lst.remove(min(lst))
lst.remove(max(lst))
print(lst)
print(round(sum(lst)/len(lst),1))
```

## 7 交换两元素

```
def swap(a, b):
    return b, a

print(swap(1, 0)) # (0,1)
```

## 8 两两组对

```
In [18]: x,y = mgrid[0:5,0:5]

In [19]: x
Out[19]:
array([[0, 0, 0, 0, 0],
       [1, 1, 1, 1, 1],
       [2, 2, 2, 2, 2],
       [3, 3, 3, 3, 3],
       [4, 4, 4, 4, 4]])

In [20]: y
Out[20]:
array([[0, 1, 2, 3, 4],
       [0, 1, 2, 3, 4],
       [0, 1, 2, 3, 4],
       [0, 1, 2, 3, 4],
       [0, 1, 2, 3, 4]])
```

这是基本用法，完全看不出干啥。如果我有10个点，想要得出这10个点的两两间距离：

```
x,y = mgrid[0:5,0:5]
In [28]: list(map(lambda xe,ye: [(ex,ey) for ex, ey in
zip(xe, ye)], x,y))
Out[28]:
[[ (0, 0), (0, 1), (0, 2), (0, 3), (0, 4)],
 [ (1, 0), (1, 1), (1, 2), (1, 3), (1, 4)],
 [ (2, 0), (2, 1), (2, 2), (2, 3), (2, 4)],
 [ (3, 0), (3, 1), (3, 2), (3, 3), (3, 4)],
 [ (4, 0), (4, 1), (4, 2), (4, 3), (4, 4)]]
```

## 9 打印99乘法表

打印出如下格式的乘法表

```
1*1=1
1*2=2    2*2=4
1*3=3    2*3=6    3*3=9
1*4=4    2*4=8    3*4=12    4*4=16
1*5=5    2*5=10    3*5=15    4*5=20    5*5=25
1*6=6    2*6=12    3*6=18    4*6=24    5*6=30    6*6=36
1*7=7    2*7=14    3*7=21    4*7=28    5*7=35    6*7=42    7*7=49
1*8=8    2*8=16    3*8=24    4*8=32    5*8=40    6*8=48    7*8=56
    8*8=64
1*9=9    2*9=18    3*9=27    4*9=36    5*9=45    6*9=54    7*9=63
    8*9=72    9*9=81
```

一共有10行，第*i*行的第*j*列等于： $j*i$ ，

其中，

*i* 取值范围： $1 \leq i \leq 9$

*j* 取值范围： $1 \leq j \leq i$

根据 例子分析 的语言描述，转化为如下代码：

```
for i in range(1,10):
    ...:     for j in range(1,i+1):
    ...:         print(str(j) + str("*") + str(i)+"=" +
str(i*j),end="\t")
    ...:     print()
```

注意 `print(str(j) + str("*") + str(i)+"=" + str(i*j),end="\t")`，两种更友好的写法：

1)

```
for i in range(1,10):
    ...:     for j in range(1,i+1):
    ...:         print('%d*d=%d'%(j,i,j*i),end="\t")
    ...:     print()
```

2)

```
for i in range(1,10):
    for j in range(1,i+1):
        print('{0}*{1}={2}'.format(j,i,j*i),end="\t")
    print()
```

## 11 嵌套数组完全展开

对于如下数组：

```
[[[1,2,3],[4,5]]]
```

如何完全展开成一维的。这个小例子实现的 `flatten` 是递归版，两个参数分别表示带展开的数组，输出数组。

```
from collections.abc import *

# 返回list
def flatten(input_arr, output_arr=None):
    if output_arr is None:
        output_arr = []
    for ele in input_arr:
        if isinstance(ele, Iterable): # 判断ele是否可迭代
            flatten(ele, output_arr) # 尾数递归
        else:
            output_arr.append(ele) # 产生结果
    return output_arr
```

调用 `flatten`：

```
print(flatten([[1,2,3],[4,5]]))
print(flatten([[1,2,3],[4,5]], [6,7]))
print(flatten([[[1,2,3],[4,5,6]]]))
# 结果:
[1, 2, 3, 4, 5]
[6, 7, 1, 2, 3, 4, 5]
[1, 2, 3, 4, 5, 6]
```

`numpy`里的 `flatten` 与上面的函数实现有些微妙的不同：

```
import numpy
b = numpy.array([[1,2,3],[4,5]])
b.flatten()
array([list([1, 2, 3]), list([4, 5])], dtype=object)
```

## 12 列表等分

```
from math import ceil

def divide(lst, size):
    if size <= 0:
        return [lst]
    return [lst[i * size:(i+1)*size] for i in range(0,
ceil(len(lst) / size))]

r = divide([1, 3, 5, 7, 9], 2)
print(r) # [[1, 3], [5, 7], [9]]

r = divide([1, 3, 5, 7, 9], 0)
print(r) # [[1, 3, 5, 7, 9]]

r = divide([1, 3, 5, 7, 9], -3)
print(r) # [[1, 3, 5, 7, 9]]
```

## 13 斐波那契数列前n项

```
def fibonacci(n):
    a, b = 1, 1
    for _ in range(n):
        yield a
        a, b = b, a + b

list(fibonacci(5)) # [1, 1, 2, 3, 5]
```

## 14 过滤掉空值

```
def filter_false(lst):
    return list(filter(bool, lst))

r = filter_false([None, 0, False, '', [], 'ok', [1, 2]])
print(r) # ['ok', [1, 2]]
```

## 15 返回更长列表

```
def max_length(*lst):
    return max(*lst, key=lambda v: len(v))

r = max_length([1, 2, 3], [4, 5, 6, 7], [8])
print(f'更长的列表是{r}') # [4, 5, 6, 7]

r = max_length([1, 2, 3], [4, 5, 6, 7], [8, 9])
print(f'更长的列表是{r}') # [4, 5, 6, 7]
```

## 16 出现次数最多的元素

```
def max_frequency(lst):
    return max(lst, default='列表为空', key=lambda v:
lst.count(v))

lst = [1, 3, 3, 2, 1, 1, 2]
r = max_frequency(lst)
print(f'{lst}中出现次数最多的元素为:{r}') # [1, 3, 3, 2, 1,
1, 2]中出现次数最多的元素为:1
```

## 17 多个列表的最大值

```
def max_lists(*lst):
    return max(max(*lst, key=lambda v: max(v)))

r = max_lists([1, 2, 3], [6, 7, 8], [4, 5])
print(r) # 8
```

## 18 检查list是否有重复元素

```
def has_duplicates(lst):
    return len(lst) == len(set(lst))

x = [1, 1, 2, 2, 3, 2, 3, 4, 5, 6]
y = [1, 2, 3, 4, 5]
has_duplicates(x) # False
has_duplicates(y) # True
```

## 19 多个列表的最小值

```
def min_lists(*lst):
    return min(min(*lst, key=lambda v: max(v)))

r = min_lists([1, 2, 3], [6, 7, 8], [4, 5])
print(r) # 1
```

## 20 找出 所有重复元素

```
from collections import Counter

def find_all_duplicates(lst):
    c = Counter(lst)
    return list(filter(lambda k: c[k] > 1, c))

find_all_duplicates([1, 2, 2, 3, 3, 3]) # [2,3]
```

## 21 列表反转

```
def reverse(lst):
    return lst[::-1]

r = reverse([1, -2, 3, 4, 1, 2])
print(r) # [2, 1, 4, 3, -2, 1]
```

## 22 浮点数等差数列

```
def rang(start, stop, n):
    start, stop, n = float('%.2f' % start), float('%.2f' % stop), int('%d' % n)
    step = (stop - start) / n
    lst = [start]
    while n > 0:
        start, n = start + step, n - 1
        lst.append(round((start), 2))
    return lst

rang(1, 8, 10) # [1.0, 1.7, 2.4, 3.1, 3.8, 4.5, 5.2, 5.9, 6.6, 7.3, 8.0]
```

## 23 按条件分组

```
def bif_by(lst, Fn):
    return [ [x for x in lst if Fn(x)], [x for x in lst if
not Fn(x)]]

records = [25,89,31,34]
bif_by(records, lambda x: x<80) # [[25, 31, 34], [89]]
```

## 24 map实现向量运算

```
#多序列运算函数-map(function,iterabel,iterable2)
lst1=[1,2,3,4,5,6]
lst2=[3,4,5,6,3,2]
list(map(lambda x,y:x*y+1,lst1,lst2))
### [4, 9, 16, 25, 16, 13]
```

## 25 联合统计次数

Counter对象间可以做数学运算

```
from collections import Counter
a = ['apple', 'orange', 'computer', 'orange']
b = ['computer', 'orange']

ca = Counter(a)
cb = Counter(b)
#Counter对象间可以做数学运算
ca + cb # Counter({'orange': 3, 'computer': 2, 'apple':
1})

# 进一步抽象，实现多个列表内元素的个数统计

def sumc(*c):
    if (len(c) < 1):
        return
    mapc = map(Counter, c)
    s = Counter([])
    for ic in mapc: # ic 是一个Counter对象
        s += ic
    return s

#Counter({'orange': 3, 'computer': 3, 'apple': 1, 'abc':
1, 'face': 1})
sumc(a, b, ['abc'], ['face', 'computer'])
```

## 26 值最大的键值对

```
def max_pairs(dic):
    if len(dic) == 0:
        return dic
    max_val = max(map(lambda v: v[1], dic.items()))
    return [item for item in dic.items() if item[1] ==
max_val]

r = max_pairs({'a': -10, 'b': 5, 'c': 3, 'd': 5})
print(r) # [('b', 5), ('d', 5)]
```

## 27 合并两个字典

```
def merge_dict2(dic1, dic2):
    return {**dic1, **dic2} # python3.5后支持的一行代码实现
合并字典

merge_dict({'a': 1, 'b': 2}, {'c': 3}) # {'a': 1, 'b': 2,
'c': 3}
```

## 28 topN最大值字典

```
from heapq import nlargest

# 返回字典d前n个最大值对应的键

def topn_dict(d, n):
    return nlargest(n, d, key=lambda k: d[k])

topn_dict({'a': 10, 'b': 8, 'c': 9, 'd': 10}, 3) # ['a',
'd', 'c']
```

## 29 最小字典

```
#求字典最小键值对
d={'a':-10,'b':5, 'c':3,'d':5}
min(d.items(),key=lambda x:x[1]) #('a', -10)
```

## 30 异位词



```

from collections import Counter

# 检查两个字符串是否 相同字母异序词，简称：互为变位词

def anagram(str1, str2):
    return Counter(str1) == Counter(str2)

anagram('eleven+two', 'twelve+one') # True 这是一对神器的变
位词
anagram('eleven', 'twelve') # False

```

### 31 反转字符串

```

#反转字符串
st="python"
#方法1
''.join(reversed(st))
#方法2
st[::-1]

```

### 32 字符串切片操作

```

字符串切片操作——查找替换3或5的倍数
" ".join([str("java"[i%3*4:]+"python"[i%5*6:] or i) for i
in range(1,15)])
'1 2 java 4 python java 7 8 java python 11 java 13 14'

```

### 33 groupby单字段分组

天气记录：

```

a = [{'date': '2019-12-15', 'weather': 'cloud'},
      {'date': '2019-12-13', 'weather': 'sunny'},
      {'date': '2019-12-14', 'weather': 'cloud'}]

```

按照天气字段weather分组汇总：

```

from itertools import groupby
for k, items in groupby(a, key=lambda x: x['weather']):
    print(k)

```

输出结果看出，分组失败！原因：分组前必须按照分组字段排序，这个很坑~

```
cloud
sunny
cloud
```

修改代码：

```
a.sort(key=lambda x: x['weather'])
for k, items in groupby(a, key=lambda x: x['weather']):
    print(k)
    for i in items:
        print(i)
```

输出结果：

```
cloud
{'date': '2019-12-15', 'weather': 'cloud'}
{'date': '2019-12-14', 'weather': 'cloud'}
sunny
{'date': '2019-12-13', 'weather': 'sunny'}
```

### 34 itemgetter和key函数

注意到 `sort` 和 `groupby` 所用的 `key` 函数，除了 `lambda` 写法外，还有一种简写，就是使用 `itemgetter`：

```
a = [{'date': '2019-12-15', 'weather': 'cloud'},
      {'date': '2019-12-13', 'weather': 'sunny'},
      {'date': '2019-12-14', 'weather': 'cloud'}]
from operator import itemgetter
from itertools import groupby

a.sort(key=itemgetter('weather'))
for k, items in groupby(a, key=itemgetter('weather')):
    print(k)
    for i in items:
        print(i)
```

结果：

```
cloud
{'date': '2019-12-15', 'weather': 'cloud'}
{'date': '2019-12-14', 'weather': 'cloud'}
sunny
{'date': '2019-12-13', 'weather': 'sunny'}
```

### 35 groupby多字段分组

`itemgetter`是一个类，`itemgetter('weather')`返回一个可调用的对象，它的参数可有多个：

```
from operator import itemgetter
from itertools import groupby

a.sort(key=itemgetter('weather', 'date'))
for k, items in groupby(a, key=itemgetter('weather')):
    print(k)
    for i in items:
        print(i)
```

结果如下，使用 `weather` 和 `date` 两个字段排序 `a`，

```
cloud
{'date': '2019-12-14', 'weather': 'cloud'}
{'date': '2019-12-15', 'weather': 'cloud'}
sunny
{'date': '2019-12-13', 'weather': 'sunny'}
```

注意这个结果与上面结果有些微妙不同，这个更多是我们想看到和使用更多的。

### 36 sum函数计算和聚合同时做

Python中的聚合类函数 `sum`, `min`, `max` 第一个参数是 `iterable` 类型，一般使用方法如下：

```
a = [4,2,5,1]
sum([i+1 for i in a]) # 16
```

使用列表生成式 `[i+1 for i in a]` 创建一个长度与 `a` 一行的临时列表，这步完成后，再做 `sum` 聚合。

试想如果你的数组 `a` 长度十百万级，再创建一个这样的临时列表就很不划算，最好是一边算一边聚合，稍改动为如下：

```
a = [4,2,5,1]
sum(i+1 for i in a) # 16
```

此时 `i+1 for i in a` 是 `(i+1 for i in a)` 的简写，得到一个生成器 (`generator`) 对象，如下所示：

```
In [8]:(i+1 for i in a)
OUT [8]:<generator object <genexpr> at 0x000002AC7FFA8CF0>
```

生成器每迭代一步吐出 (`yield`) 一个元素并计算和聚合后，进入下一次迭代，直到终点。

## 四、Python之能

Python之能更多是实战中的28个小功能，它比Python之例中的例子代码行数多一些。在教会你搭建好Python环境后，一起实现这些有意思的小功能。

### 1 环境搭建

区分几个小白容易混淆的概念：pycharm，python解释器，conda安装，pip安装，总结来说：

- **pycharm**是python开发的集成开发环境(Integrated Development Environment，简称IDE)，它本身无法执行Python代码
- **python解释器**才是真正执行代码的工具，pycharm里可设置Python解释器，一般我们可去python官网下载python3.7或python3.8版本；如果安装过**anaconda**，它里面必然也包括一个某版本的Python解释器：pycharm配置python解释器选择哪一个都可以。
- **anaconda**是把python所有常用包的合集，并提供给我们使用**conda**命令非常非常方便的安装各种Python包。
- **conda安装**：我们安装过anaconda软件后，就能够使用conda命令下载anaconda源里(比如中科大镜像源)的包
- **pip安装**：也是一种类似于conda安装的python安装方法，如果用过Centos系统，它就像**yum**安装一样。

#### 修改镜像源

在使用安装**conda**安装某些包会出现慢或安装失败问题，最有效方法是修改镜像源为国内镜像源。之前都选用清华镜像源，但是2019年后已停止服务。推荐选用中科大镜像源。

先查看已经安装过的镜像源，cmd窗口执行命令：

```
conda config --show
```

查看配置项**channels**，如果显示带有**tsinghua**，则说明已安装过清华镜像。

```
channels:
-
https://mirrors.tuna.tsinghua.edu.cn/tensorflow/linux/cpu/
-
https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/msys2/
-
https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda-
forge/
- https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkg/free/
-
https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/pytorc
h/
```

下一步，使用 `conda config --remove channels url地址` 删除清华镜像，如下命令删除第一个。然后，依次删除所有镜像源

```
conda config --remove channels
https://mirrors.tuna.tsinghua.edu.cn/tensorflow/linux/cpu/
```

添加目前可用的中科大镜像源：

```
conda config --add channels
https://mirrors.ustc.edu.cn/anaconda/pkg/free/
```

并设置搜索时显示通道地址：

```
conda config --set show_channel_urls yes
```

确认是否安装镜像源成功，执行 `conda config --show`，找到 `channels` 值为如下：

```
channels:
- https://mirrors.ustc.edu.cn/anaconda/pkg/free/
- defaults
```

Done~

## 2 自动群发邮件

Python自动群发邮件

```
import smtplib
from email import header
from email.mime import text, application, multipart
import time

def sender_mail():
```

```

smt_p = smtplib.SMTP()
smt_p.connect(host='smtp.qq.com', port=25)
sender, password = '113097485@qq.com',
"*****"

smt_p.login(sender, password)
receiver_addresses, count_num = [
    'guozhennianhua@163.com', 'xiaoxiazi99@163.com'],
1
for email_address in receiver_addresses:
    try:
        msg = multipart.MIMEMultipart()
        msg['From'] = "zhenguo"
        msg['To'] = email_address
        msg['subject'] = header.Header('这是邮件主题通
知', 'utf-8')
        msg.attach(text.MIMEText(
            '这是一封测试邮件，请勿回复本邮件~', 'plain',
            'utf-8'))
        smt_p.sendmail(sender, email_address,
msg.as_string())
        time.sleep(10)
        print('第%d次发送给%s' % (count_num,
email_address))
        count_num = count_num + 1
    except Exception as e:
        print('第%d次给%s发送邮件异常' % (count_num,
email_address))
        continue
    smt_p.quit()

sender_mail()

```

注意：

发送邮箱是qq邮箱，所以要在qq邮箱中设置开启SMTP服务，设置完成时会生成一个授权码，将这个授权码赋值给文中的password变量。

发送后的截图：

## 这是邮件主题通知

发件人: "zhenguo" <>> (由 113097485@qq.com 代发, 帮助)

收件人: 我<guozhennianhua@163.com>

时 间: 2019年12月03日 13:55 (星期二)

你可以用泛微OA流程审批、合同管理.... [在线试用>>](#)

这是一封测试邮件，请勿回复本邮件~

### 3 二分搜索

二分搜索是程序员必备的小算法，无论什么场合，都要非常熟练地写出来。

小例子描述：

在有序数组 `arr` 中，指定区间 `[left, right]` 范围内，查找元素 `x`

如果不存在，返回 `-1`

二分搜索 `binarySearch` 实现的主逻辑

```
def binarySearch(arr, left, right, x):
    while left <= right:

        mid = int(left + (right - left) / 2); # 找到中间位置。求中点写成(left+right)/2更容易溢出，所以不建议这样写

        # 检查x是否出现在位置mid
        if arr[mid] == x:
            print('found %d 在索引位置%d 处' %(x,mid))
            return mid

        # 假如x更大，则不可能出现在左半部分
        elif arr[mid] < x:
            left = mid + 1 #搜索区间变为[mid+1,right]
            print('区间缩小为[%d,%d]' %(mid+1,right))

        # 同理，假如x更小，则不可能出现在右半部分
        elif x<arr[mid]:
            right = mid - 1 #搜索区间变为[left,mid-1]
            print('区间缩小为[%d,%d]' %(left,mid-1))

        # 假如搜索到这里，表明x未出现在[left,right]中
    return -1
```

在 `Ipython` 交互界面中，调用 `binarySearch` 的小Demo:

```
In [8]: binarySearch([4,5,6,7,10,20,100],0,6,5)
```

```
区间缩小为[0,2]
found 5 at 1
Out[8]: 1

In [9]: binarySearch([4,5,6,7,10,20,100],0,6,4)
区间缩小为[0,2]
区间缩小为[0,0]
found 4 at 0
Out[9]: 0

In [10]: binarySearch([4,5,6,7,10,20,100],0,6,20)
区间缩小为[4,6]
found 20 at 5
Out[10]: 5

In [11]: binarySearch([4,5,6,7,10,20,100],0,6,100)
区间缩小为[4,6]
区间缩小为[6,6]
found 100 at 6
Out[11]: 6
```

## 4 批量修改文件后缀

### 批量修改文件后缀

本例子使用Python的 `os` 模块和 `argparse` 模块，将工作目录 `work_dir` 下所有后缀名为 `old_ext` 的文件修改为后缀名为 `new_ext`

通过本例子，大家将会大概清楚 `argparse` 模块的主要用法。

导入模块

```
import argparse
import os
```

定义脚本参数



```
def get_parser():
    parser = argparse.ArgumentParser(
        description='工作目录中文件后缀名修改')
    parser.add_argument('work_dir', metavar='WORK_DIR',
                        type=str, nargs=1,
                        help='修改后缀名的文件目录')
    parser.add_argument('old_ext', metavar='OLD_EXT',
                        type=str, nargs=1, help='原来的后
                        缀')
    parser.add_argument('new_ext', metavar='NEW_EXT',
                        type=str, nargs=1, help='新的后缀')
    return parser
```

后缀名批量修改

```
def batch_rename(work_dir, old_ext, new_ext):
    """
    传递当前目录，原来后缀名，新的后缀名后，批量重命名后缀
    """
    for filename in os.listdir(work_dir):
        # 获取得到文件后缀
        split_file = os.path.splitext(filename)
        file_ext = split_file[1]
        # 定位后缀名为old_ext 的文件
        if old_ext == file_ext:
            # 修改后文件的完整名称
            newfile = split_file[0] + new_ext
            # 实现重命名操作
            os.rename(
                os.path.join(work_dir, filename),
                os.path.join(work_dir, newfile)
            )
    print("完成重命名")
    print(os.listdir(work_dir))
```

实现Main

```
def main():
    """
    main函数
    """
    # 命令行参数
    parser = get_parser()
    args = vars(parser.parse_args())
    # 从命令行参数中依次解析出参数
    work_dir = args['work_dir'][0]
    old_ext = args['old_ext'][0]
    if old_ext[0] != '.':
        old_ext = '.' + old_ext
    new_ext = args['new_ext'][0]
```

```

if new_ext[0] != '.':
    new_ext = '.' + new_ext

batch_rename(work_dir, old_ext, new_ext)

```

## 5 定制文件不同行

比较两个文件在哪些行内容不同，返回这些行的编号，行号编号从1开始。

定义统计文件行数的函数

```

# 统计文件个数
def statLineCnt(statfile):
    print('文件名: ' + statfile)
    cnt = 0
    with open(statfile, encoding='utf-8') as f:
        while f.readline():
            cnt += 1
    return cnt

```

统计文件不同之处的子函数：

```

# more表示含有更多行数的文件
def diff(more, cnt, less):
    difflist = []
    with open(less, encoding='utf-8') as l:
        with open(more, encoding='utf-8') as m:
            lines = l.readlines()
            for i, line in enumerate(lines):
                if line.strip() !=
m.readline().strip():
                    difflist.append(i)
            if cnt - i > 1:
                difflist.extend(range(i + 1, cnt))
    return [no+1 for no in difflist]

```

主函数：

```

# 返回的结果行号从1开始
# list表示fileA和fileB不同的行的编号

def file_diff_line_nos(fileA, fileB):
    try:
        cntA = statLineCnt(fileA)
        cntB = statLineCnt(fileB)
        if cntA > cntB:
            return diff(fileA, cntA, fileB)
        return diff(fileB, cntB, fileA)

    except Exception as e:
        print(e)

```

比较两个文件A和B，拿相对较短的文件去比较，过滤行后的换行符\n和空格。

暂未考虑某个文件最后可能有的多行空行等特殊情况

使用 `file_diff_line_nos` 函数：

```

if __name__ == '__main__':
    import os
    print(os.getcwd())

    '''
    例子：
    fileA = "'hello world!!!!'\n
            'nice to meet you'\n
            'yes'\n
            'no1'\n
            'jack'"
    fileB = "'hello world!!!!'\n
            'nice to meet you'\n
            'yes' "
    '''

    diff = file_diff_line_nos('./testdir/a.txt',
                              './testdir/b.txt')
    print(diff)  # [4, 5]

```

关于文件比较的，实际上，在Python中有对应模块 `difflib`，提供更多其他格式的文件更详细的比较，大家可参考：

<https://docs.python.org/3/library/difflib.html?highlight=difflib#module-difflib>

## 6 指定后缀名的文件

```
import os
```

```

def find_file(work_dir,extension='jpg'):
    lst = []
    for filename in os.listdir(work_dir):
        print(filename)
        splits = os.path.splitext(filename)
        ext = splits[1] # 拿到扩展名
        if ext == '.'+extension:
            lst.append(filename)
    return lst

r = find_file('.', 'md')
print(r) # 返回所有目录下的md文件

```

## 7 xls批量转换成xlsx

```

#批量转换文件xls-xlsx
import win32com.client as win32
import os.path
import os

def xls2xlsx():
    rootdir = r"C:\Users\CQ375\Desktop\temp1" #需要转换的
    xls文件存放处
    rootdir1 = r"C:\Users\CQ375\Desktop\ex" #转换好的xlsx文
    件存放处
    files = os.listdir(rootdir) #列出xls文件夹下的所有文件

    num = len(files) #列出所有文件的个数
    for i in range(num): #按文件个数执行次数
        kname = os.path.splitext(files[i])[1] #分离文件名与
        扩展名, 返回(f_name, f_extension)元组
        if kname == '.xls': #判定扩展名是否为xls,屏蔽其它文件

            fname = rootdir + '\\' + files[i] #合成需要转换
            的路径与文件名
            fname1 = rootdir1 + '\\' + files[i] #合成准备存
            放转换好的路径与文件名
            excel =
            win32.gencache.EnsureDispatch('Excel.Application') #调用
            win32模块

            wb = excel.workbooks.Open(fname) #打开需要转换的
            文件

            wb.SaveAs(fname1+"x", FileFormat=51) #文件另存
            为xlsx扩展名的文件

```

```

wb.close()
excel.Application.Quit()

if __name__ == '__main__':
    xls2xlsx()

```

## 8 批量获取文件修改时间

```

#获取目录下文件的修改时间
import os
import datetime
print(f"当前时间: {datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")
for root,dirs,files in os.walk(r"D:\works"): #循环D:\works目录和子目录
    for file in files:
        absPathFile=os.path.join(root,file)

        modiefiedTime=datetime.datetime.fromtimestamp(os.path.getmtime(absPathFile))
        now=datetime.datetime.now()
        diffTime=now-modiefiedTime
        if diffTime.days<20: #条件筛选超过指定时间的文件
            print(f"{absPathFile:<27s}修改时间 [{modiefiedTime.strftime('%Y-%m-%d %H:%M:%S')}] \
距今[{diffTime.days:3d}天{diffTime.seconds//3600:2d}时 {diffTime.seconds%3600//60:2d}]") #打印相关信息

```

```

当前时间: 2019-12-07 22:06:43
D:\works\工作资料目录\设备信息变更表2019.11.8.xlsx修改时间[2019-11-19 23:19:06]距今[ 17天22时47]
D:\works\工作资料目录\建筑基础资料整理\XML支路20191119.xlsx修改时间[2019-11-19 23:57:15]距今[ 17天22时 9]
D:\works\工作资料目录\建筑基础资料整理\XML支路20191121.xlsx修改时间[2019-11-20 22:43:36]距今[ 16天23时23]

```

## 9 日期计算

```

#计算指定日期当月最后一天的日期和该月天数
import datetime
import calendar
init_date = datetime.date.today()
print('当前给定时间:', init_date)
current_month_days=calendar.monthrange(init_date.year,init_date.month)[1]
print(calendar.month(2019,init_date.month))
current_month_last_day = datetime.date(init_date.year,init_date.month, current_month_days)
print("当月最后一天:", current_month_last_day)
print("该月天数:", current_month_days)

```

```

当前给定时间: 2019-12-08
      December 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

当月最后一天: 2019-12-31
该月天数: 31

```

## 10 批量压缩文件

```

import zipfile # 导入zipfile,这个是用来做压缩和解压的Python模块;
import os
import time

def batch_zip(start_dir):
    start_dir = start_dir # 要压缩的文件夹路径
    file_news = start_dir + '.zip' # 压缩后文件夹的名字

    z = zipfile.ZipFile(file_news, 'w',
zipfile.ZIP_DEFLATED)
    for dir_path, dir_names, file_names in
os.walk(start_dir):
        # 这一句很重要,不replace的话,就从根目录开始复制
        f_path = dir_path.replace(start_dir, '')
        f_path = f_path and f_path + os.sep # 实现当前文件
        夹以及包含的所有文件的压缩
        for filename in file_names:
            z.write(os.path.join(dir_path, filename),
f_path + filename)
    z.close()

```

```
        return file_news

batch_zip('./data/ziptest')
```

## 11 文件读写

```
import os
# 创建文件夹

def mkdir(path):
    isexists = os.path.exists(path)
    if not isexists:
        os.mkdir(path)
# 读取文件信息

def openfile(filename):
    f = open(filename)
    flist = f.read()
    f.close()
    return flist # 返回读取内容

# 写入文件信息

# example1
# w写入，如果文件存在，则清空内容后写入，不存在则创建
f = open(r"./data/test.txt", "w", encoding="utf-8")
print(f.write("测试文件写入"))
f.close()
# example2
# a写入，文件存在，则在文件内容后追加写入，不存在则创建
f = open(r"./data/test.txt", "a", encoding="utf-8")
print(f.write("测试文件写入"))
f.close()

# example3
# with关键字系统会自动关闭文件和处理异常
with open(r"./data/test.txt", "w") as f:
    f.write("hello world!")
```

## 12 32位加密

```
import hashlib
# 对字符串s实现32位加密

def hash_cry32(s):
    m = hashlib.md5()
    m.update((str(s).encode('utf-8')))
    return m.hexdigest()

print(hash_cry32(1)) # c4ca4238a0b923820dcc509a6f75849b
print(hash_cry32('hello')) #
5d41402abc4b2a76b9719d911017c592
```

### 13 密码合法性判断

Python的`re`模块提供字符串正则匹配检查，功能强大，写法高效简洁，因此在工作中会被经常使用。

今天举一个小例子说明`re`的一些主要功能。

例子描述

判断密码是否安全，设计一个密码是否安全的检查函数。

密码安全要求：

- 要求密码为6到20位，
- 密码只包含英文字母和数字

导入模块

导入正则模块

```
import re
```

编写正则规则

```
a = re.compile(r'[0-9a-zA-Z]{6,20}')
```

`a`为正则对象，里面方法包括`match`、`fullmatch`等

`r`表示后面为正则字符

`[]`表示匹配字符集合，此处`0-9a-zA-Z`满足 密码只包含英文字母和数字

`{6,20}`表示字符长度，满足要求 密码为6到20位

检查例子



```
In [6]: a.fullmatch('ddd234ws')
Out[6]: Match object; span=(0, 8), match='ddd234ws'>
```

fullmatch表示整个字符串是否匹配，显然字符串 `ddd234ws` 完全匹配

```
In [7]: a.fullmatch('ddd234ws###')
# 返回None，表示字符串不匹配我们的要求

# 如下都是不匹配的例子
In [8]: a.fullmatch('dd')
In [9]: a.fullmatch('dd3')
In [10]:
a.fullmatch('dd3wsxfsdfsfsfdwe3342532fscsdcsdfsdfs')

```

完整代码

```
import re
def check(mystr):
    a = re.compile(r'[0-9a-zA-Z]{6,20}')
    if a.fullmatch(mystr) is None:
        return '密码只能包含英文字母和数字，长度6~20'
    return '密码安全'
```

## 14 使用正则批量转化为驼峰

数据库字段名批量转化为驼峰格式

分析过程

```
# 用到的正则串讲解
# \s 指匹配: [ \t\n\r\f\v]
# A|B: 表示匹配A串或B串
# re.sub(pattern, newchar, string):
# substitute代替，用newchar字符替代与pattern匹配的字符所有。
```

```
# title(): 转化为大写，例子：
# 'Hello world'.title() # 'Hello world'
```

```
# print(re.sub(r"\s|_|-", "", "He llo_worl\td"))
s = re.sub(r"(\s|_|-)+", " ",
           'some_database_field_name').title().replace(" ", "")
#结果:  SomeDatabaseFieldName
```

```
# 可以看到此时的第一个字符为大写，需要转化为小写
s = s[0].lower()+s[1:] # 最终结果
```

整理以上分析得到如下代码：

```
import re
def camel(s):
    s = re.sub(r"(\s|_|-)+", " ", s).title().replace(" ", "")
    return s[0].lower() + s[1:]

# 批量转化
def batch_camel(slist):
    return [camel(s) for s in slist]
```

测试结果：

```
s = batch_camel(['student_id', 'student\tname', 'student-
add'])
print(s)
# 结果
['studentId', 'studentName', 'studentAdd']
```

## 15 爬取天气数据并解析温度值

爬取天气数据并解析温度值

素材来自朋友袁绍，感谢！

爬取的html 结构

```

    <div class="tq_zx" id="tq_zx">...</div>
    <div class="left-div">...</div>
    <div id="around" class="around">
      <div class="aro_city" style="display:block;">
        <input type="hidden" id="around_city_china_update_time" value="2019111708">
        <h1 class="clearfix city">...</h1>
        <ul class="clearfix city">
          <li>
            <a href="http://www.weather.com.cn/weather1d/101090604.shtml#around2" target=
              "_blank">...</a>
          </li>
          <li>
            <a href="http://www.weather.com.cn/weather1d/101090218.shtml#around2" target=
              "_blank">...</a>
          </li>
          <li>
            <a href="http://www.weather.com.cn/weather1d/101090501.shtml#around2" target=
              "_blank">...</a>
          </li>
          <li>
            <a href="http://www.weather.com.cn/weather1d/101090701.shtml#around2" target=
              "_blank">
              <span>沧州</span>
              <p class="img clearfix">...</p>
              <i>14/-5°C</i> == $0
            </a>
          </li>
          <li>
            <a href="http://www.weather.com.cn/weather1d/101030100.shtml#around2" target=
              "_blank">
              <span>天津</span>
              <p class="img clearfix">...</p>
              <i>12/-1°C</i>
            </a>
          </li>
        </ul>
      </div>
    </div>
  </li>

```

```

import requests
from lxml import etree
import pandas as pd
import re

url =
'http://www.weather.com.cn/weather1d/101010100.shtml#input
'

with requests.get(url) as res:
    content = res.content
    html = etree.HTML(content)

```

通过lxml模块提取值

lxml比beautifulsoup解析在某些场合更高效

```

location = html.xpath('//*
[@id="around"]//a[@target="_blank"]/span/text()')
temperature = html.xpath('//*
[@id="around"]/div/ul/li/a/i/text()')

```

结果:

```
['香河', '涿州', '唐山', '沧州', '天津', '廊坊', '太原', '石家庄', '涿鹿', '张家口', '保定', '三河', '北京孔庙', '北京国子监', '中国地质博物馆', '月坛公园', '明城墙遗址公园', '北京市规划展览馆', '什刹海', '南锣鼓巷', '天坛公园', '北海公园', '景山公园', '北京海洋馆']

['11/-5°C', '14/-5°C', '12/-6°C', '12/-5°C', '11/-1°C', '11/-5°C', '8/-7°C', '13/-2°C', '8/-6°C', '5/-9°C', '14/-6°C', '11/-4°C', '13/-3°C', '13/-3°C', '12/-3°C', '12/-3°C', '13/-3°C', '12/-2°C', '12/-3°C', '13/-3°C', '12/-2°C', '12/-2°C', '12/-2°C', '12/-3°C']
```

构造DataFrame对象

```
df = pd.DataFrame({'location':location,
'temperature':temperature})
print('温度列')
print(df['temperature'])
```

正则解析温度值

```
df['high'] = df['temperature'].apply(lambda x:
int(re.match('(-?[0-9]*?)/-?[0-9]*?°C', x).group(1) ) )
df['low'] = df['temperature'].apply(lambda x:
int(re.match('(-?[0-9]*?)/(-?[0-9]*?)°C', x).group(1) ) )
print(df)
```

详细说明子字符创捕获

除了简单地判断是否匹配之外，正则表达式还有提取子串的强大功能。用 `()` 表示的就是要提取的分组（group）。比如：`^(\d{3})-(\d{3,8})$` 分别定义了两个组，可以直接从匹配的字符串中提取出区号和本地号码

```
m = re.match(r'^(\d{3})-(\d{3,8})$', '010-12345')
print(m.group(0))
print(m.group(1))
print(m.group(2))

# 010-12345
# 010
# 12345
```

如果正则表达式中定义了组，就可以在 `Match` 对象上用 `group()` 方法提取出子串来。

注意到 `group(0)` 永远是原始字符串，`group(1)`、`group(2)`.....表示第1、2、.....个子串。

最终结果

```
Name: temperature, dtype: object
  location temperature high low
0      香河      11/-5°C    11  -5
1      涿州      14/-5°C    14  -5
2      唐山      12/-6°C    12  -6
3      沧州      12/-5°C    12  -5
4      天津      11/-1°C    11  -1
5      廊坊      11/-5°C    11  -5
6      太原       8/-7°C     8  -7
7    石家庄      13/-2°C    13  -2
8      涿鹿       8/-6°C     8  -6
9    张家口       5/-9°C     5  -9
10     保定      14/-6°C    14  -6
11     三河      11/-4°C    11  -4
12  北京孔庙      13/-3°C    13  -3
13  北京国子监      13/-3°C    13  -3
14  中国地质博物馆      12/-3°C    12  -3
15     月坛公园      12/-3°C    12  -3
16  明城墙遗址公园      13/-3°C    13  -3
17  北京市规划展览馆      12/-2°C    12  -2
18     什刹海      12/-3°C    12  -3
19  南锣鼓巷      13/-3°C    13  -3
20     天坛公园      12/-2°C    12  -2
21     北海公园      12/-2°C    12  -2
22     景山公园      12/-2°C    12  -2
23  北京海洋馆      12/-3°C    12  -3
```

## 16 定制递减迭代器

#编写一个迭代器，通过循环语句，实现对某个正整数的依次递减1，直到0。

```
class Descend(Iterator):
    def __init__(self,N):
        self.N=N
        self.a=0
    def __iter__(self):
        return self
    def __next__(self):
        while self.a<self.N:
            self.N-=1
            return self.N
        raise StopIteration

descend_iter=Descend(10)
print(list(descend_iter))
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

核心要点：

1 `__next__` 名字不能变，实现定制的迭代逻辑

2 `raise StopIteration`: 通过 `raise` 中断程序，必须这样写

## 17 测试运行时长的装饰器

```
#测试函数执行时间的装饰器示例
import time
def timing_func(fn):
    def wrapper():
        start=time.time()
        fn()    #执行传入的fn参数
        stop=time.time()
        return (stop-start)
    return wrapper
@timing_func
def test_list_append():
    lst=[]
    for i in range(0,100000):
        lst.append(i)
@timing_func
def test_list_compre():
    [i for i in range(0,100000)]    #列表生成式
a=test_list_append()
c=test_list_compre()
print("test list append time:",a)
print("test list comprehension time:",c)
print("append/compre:",round(a/c,3))

test list append time: 0.0219423770904541
test list comprehension time: 0.007980823516845703
append/compre: 2.749
```

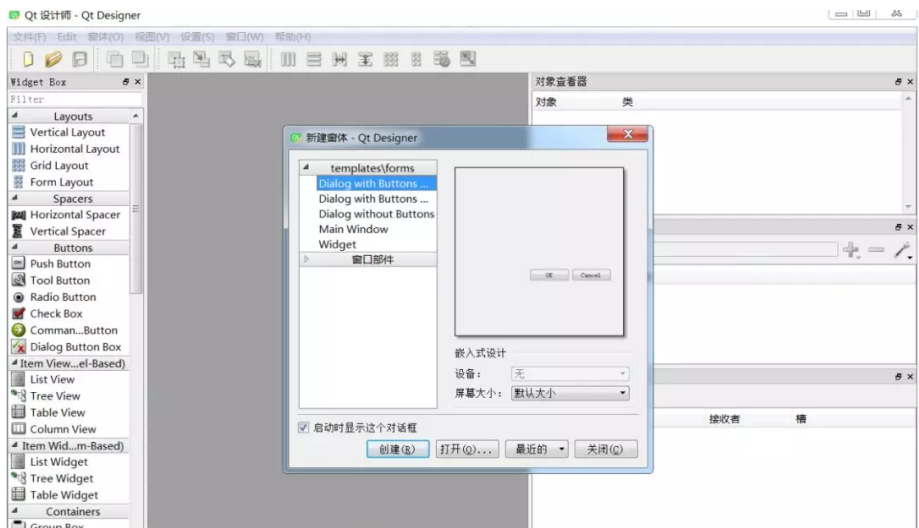
## 18 制作小而美的计算器

1) ui设计

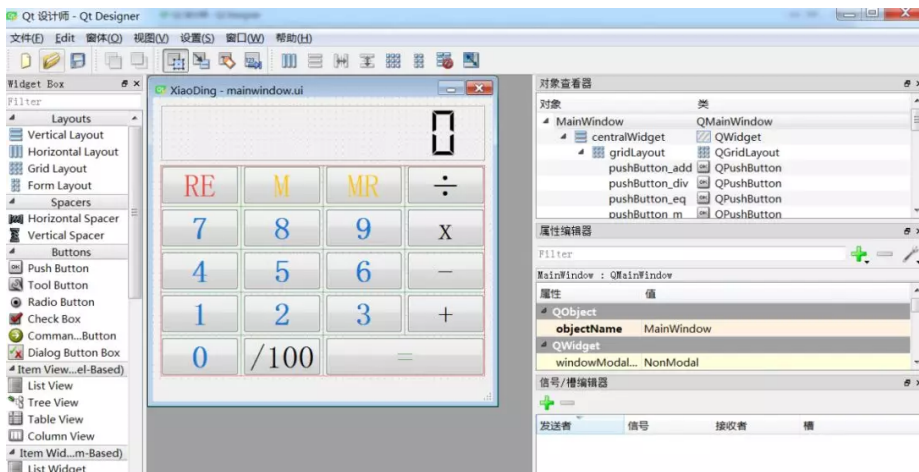
使用 `qt designer`，按装anaconda后，在如下路径找到：

```
conda3.05\Library\bin
```

`designer.exe`文件，双击启动：



创建窗体，命名为XiaoDing，整个的界面如下所示：

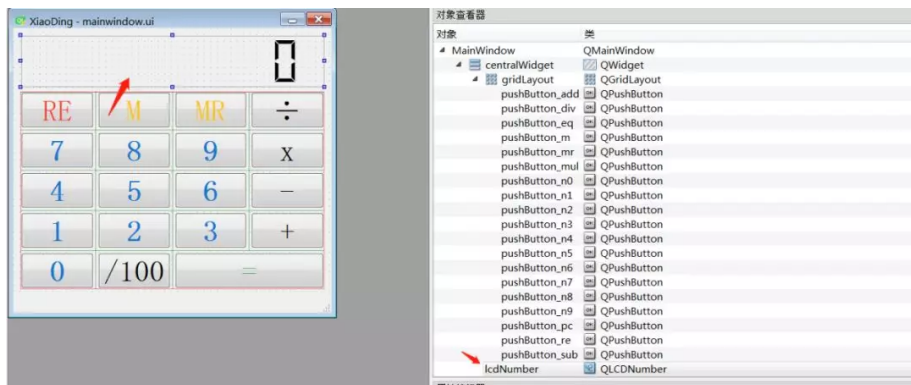


qt 设计器 提供的常用控件基本都能满足开发需求，通过拖动左侧的控件，很便捷的就能搭建出如下的UI界面，比传统的手写控件代码要方便很多。

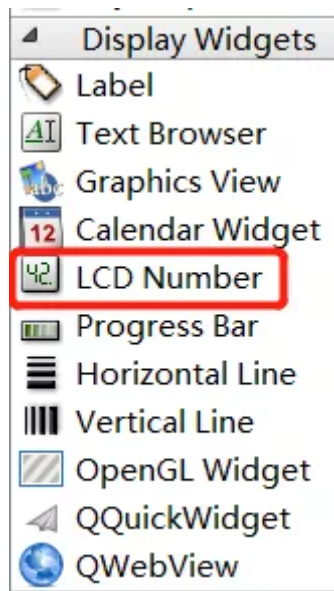
最终设计的计算器XiaoDing界面如下，



比如，其中一个用于计算器显示的对象：`lcdNumber`，对象的类型为：`LCDNumber`。右侧为计算器中用到的所有对象。







## 2) 转py文件

使用如下命令，将设计好的ui文件转为py文件：

```
pyuic5 -o ./calculator/Mainwindow.py  
./calculator/mainwindow.ui
```

## 3) 计算器实现逻辑

导入库：

```
from PyQt5.QtGui import *  
from PyQt5.QtWidgets import *  
from PyQt5.QtCore import *  
  
import operator  
  
from Mainwindow import Ui_Mainwindow
```

主题代码逻辑很精简：

```
# Calculator state.  
READY = 0  
INPUT = 1  
  
class MainWindow(QMainWindow, Ui_Mainwindow):  
    def __init__(self, *args, **kwargs):  
        super(MainWindow, self).__init__(*args, **kwargs)  
        self.setupUi(self)  
  
        # Setup numbers.  
        for n in range(0, 10):
```

```

        getattr(self, 'pushButton_n%s' %
n).pressed.connect(lambda v=n: self.input_number(v))

        # Setup operations.
        self.pushButton_add.pressed.connect(lambda:
self.operation(operator.add))
        self.pushButton_sub.pressed.connect(lambda:
self.operation(operator.sub))
        self.pushButton_mul.pressed.connect(lambda:
self.operation(operator.mul))
        self.pushButton_div.pressed.connect(lambda:
self.operation(operator.truediv)) # operator.div for
Python2.7

        self.pushButton_pc.pressed.connect(self.operation_pc)
        self.pushButton_eq.pressed.connect(self.equals)

        # Setup actions
        self.actionReset.triggered.connect(self.reset)
        self.pushButton_ac.pressed.connect(self.reset)

        self.actionExit.triggered.connect(self.close)

        self.pushButton_m.pressed.connect(self.memory_store)

        self.pushButton_mr.pressed.connect(self.memory_recall)

        self.memory = 0
        self.reset()

        self.show()

```

基础方法:

```

def input_number(self, v):
    if self.state == READY:
        self.state = INPUT
        self.stack[-1] = v
    else:
        self.stack[-1] = self.stack[-1] * 10 + v

    self.display()

def display(self):
    self.lcdNumber.display(self.stack[-1])

```

按钮 **RE**, **M**, **RE** 对应的实现逻辑:

```
def reset(self):
    self.state = READY
    self.stack = [0]
    self.last_operation = None
    self.current_op = None
    self.display()

def memory_store(self):
    self.memory = self.lcdNumber.value()

def memory_recall(self):
    self.state = INPUT
    self.stack[-1] = self.memory
    self.display()
```

**+**, **-**, **x**, **/**, **/100** 对应实现方法:

```
def operation(self, op):
    if self.current_op: # Complete the current
operation
        self.equals()

    self.stack.append(0)
    self.state = INPUT
    self.current_op = op

def operation_pc(self):
    self.state = INPUT
    self.stack[-1] *= 0.01
    self.display()
```

**=** 号对应的方法实现:

```
def equals(self):
    if self.state == READY and self.last_operation:
        s, self.current_op = self.last_operation
        self.stack.append(s)

    if self.current_op:
        self.last_operation = self.stack[-1],
self.current_op

    try:
        self.stack =
[self.current_op(*self.stack)]
    except Exception:
```

```

        self.lcdNumber.display('Err')
        self.stack = [0]
    else:
        self.current_op = None
        self.state = READY
        self.display()

```

main函数:

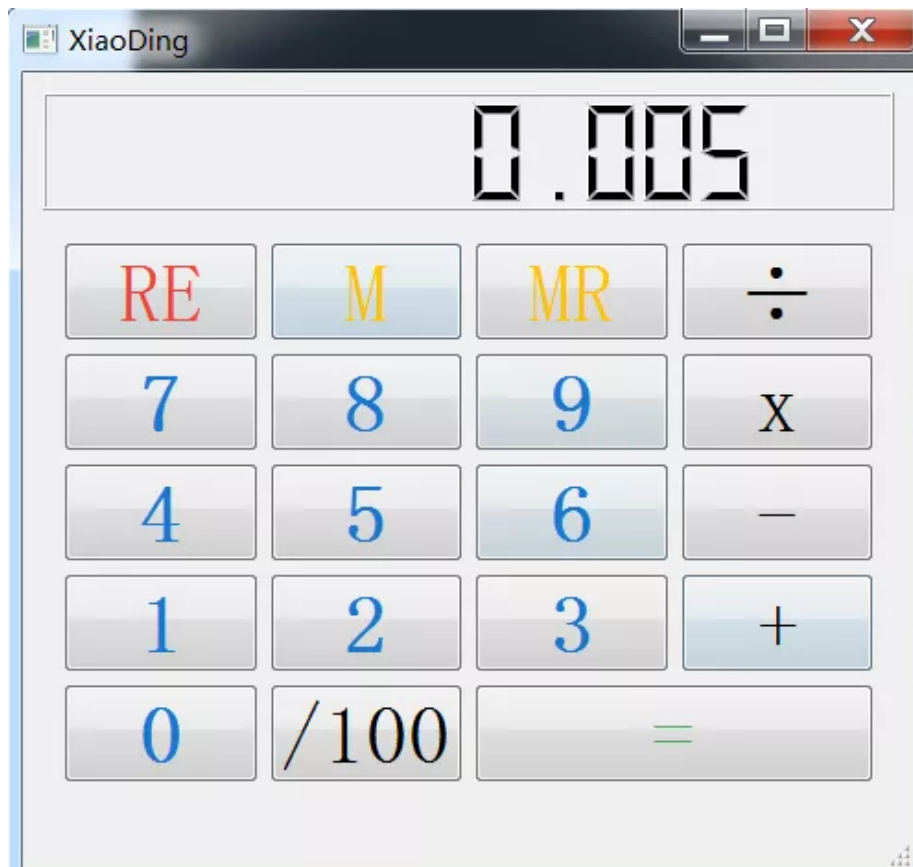
```

if __name__ == '__main__':
    app = QApplication([])
    app.setApplicationName("XiaoDing")

    window = MainWindow()
    app.exec_()

```

完整代码请参考[点击阅读原文下载](#)，代码只有100行。完整代码请点击文章最底部的【[阅读原文](#)】。启动后的界面如下：



## 19 turtle绘制奥运五环图

turtle绘图的函数非常好用，基本看到函数名字，就能知道它的含义，下面使用turtle，仅用15行代码来绘制奥运五环图。

## 1 导入库

```
import turtle
```

## 2 定义画圆函数

```
def drawCircle(x,y,c='red'):  
    p.pu()# 抬起画笔  
    p.goto(x,y) # 绘制圆的起始位置  
    p.pd()# 放下画笔  
    p.color(c)# 绘制c色圆环  
    p.circle(30,360) #绘制圆：半径，角度
```

## 3 画笔基本设置

```
p = turtle  
p.pensize(3) # 画笔尺寸设置3
```

## 4 绘制五环图

调用画圆函数

```
drawCircle(0,0,'blue')  
drawCircle(60,0,'black')  
drawCircle(120,0,'red')  
drawCircle(90,-30,'green')  
drawCircle(30,-30,'yellow')  
  
p.done()
```

结果：



## 20 turtle绘制漫天雪花

导入模块

导入 `turtle` 库和 python 的 `random`

```
import turtle as p
import random
```

绘制雪花

```
def snow(snow_count):
    p.hideturtle()
    p.speed(500)
    p.pensize(2)
    for i in range(snow_count):
        r = random.random()
        g = random.random()
        b = random.random()
        p.pencolor(r, g, b)
        p.pu()
        p.goto(random.randint(-350, 350),
random.randint(1, 270))
        p.pd()
        dens = random.randint(8, 12)
        snowsize = random.randint(10, 14)
        for _ in range(dens):
            p.forward(snowsize) # 向当前画笔方向移动snowsize
            # 像素长度
            p.backward(snowsize) # 向当前画笔相反方向移动
            # snowsize像素长度
            p.right(360 / dens) # 顺时针移动360 / dens度
```

绘制地面

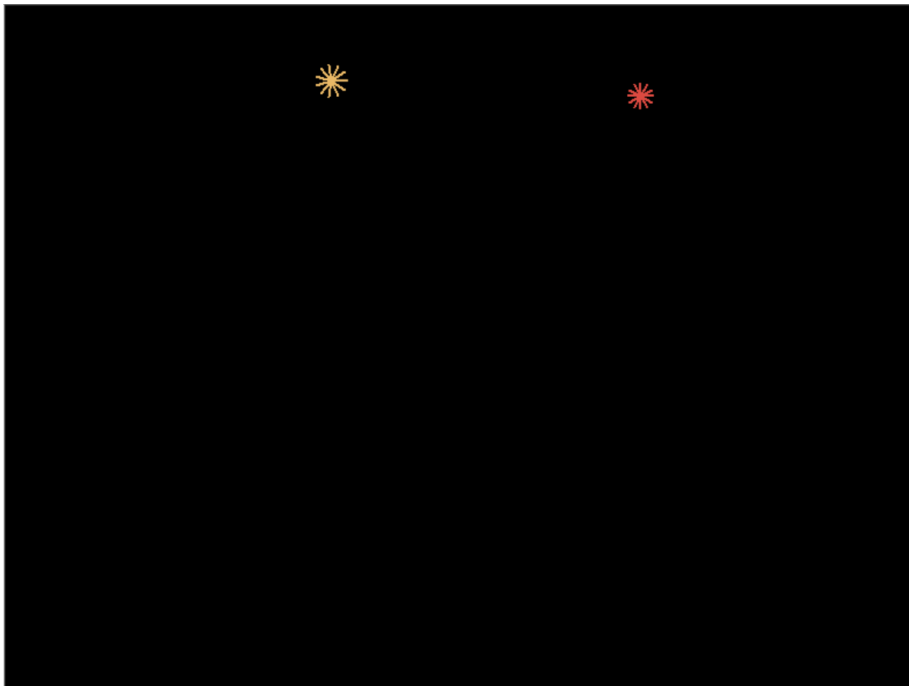
```
def ground(ground_line_count):
    p.hideturtle()
    p.speed(500)
    for i in range(ground_line_count):
        p.pensize(random.randint(5, 10))
        x = random.randint(-400, 350)
        y = random.randint(-280, -1)
        r = -y / 280
        g = -y / 280
        b = -y / 280
        p.pencolor(r, g, b)
        p.penup() # 抬起画笔
        p.goto(x, y) # 让画笔移动到此位置
        p.pendown() # 放下画笔
        p.forward(random.randint(40, 100)) # 眼当前画笔方向
        # 向前移动40~100距离
```

主函数

```
def main():
    p.setup(800, 600, 0, 0)
    # p.tracer(False)
    p.bgcolor("black")
    snow(30)
    ground(30)
    # p.tracer(True)
    p.mainloop()

main()
```

动态图结果展示：



## 21 wordcloud词云图

```
import hashlib
import pandas as pd
from wordcloud import WordCloud
geo_data=pd.read_excel(r"../data/geo_data.xlsx")
print(geo_data)
# 0      深圳
# 1      深圳
# 2      深圳
# 3      深圳
# 4      深圳
# 5      深圳
# 6      深圳
# 7      广州
# 8      广州
# 9      广州
```

```

words = ','.join(x for x in geo_data['city'] if x != []) #
筛选出非空列表值
wc = WordCloud(
    background_color="green", #背景颜色"green"绿色
    max_words=100, #显示最大词数
    font_path='./fonts/simhei.ttf', #显示中文
    min_font_size=5,
    max_font_size=100,
    width=500 #图幅宽度
)
x = wc.generate(words)
x.to_file('./data/geo_data.png')

```




## 22 plotly画柱状图和折线图

```

#柱状图+折线图
import plotly.graph_objects as go
fig = go.Figure()
fig.add_trace(
    go.Scatter(
        x=[0, 1, 2, 3, 4, 5],
        y=[1.5, 1, 1.3, 0.7, 0.8, 0.9]
    ))
fig.add_trace(
    go.Bar(
        x=[0, 1, 2, 3, 4, 5],
        y=[2, 0.5, 0.7, -1.2, 0.3, 0.4]
    ))
fig.show()

```

 1576311610044

## 23 seaborn热力图



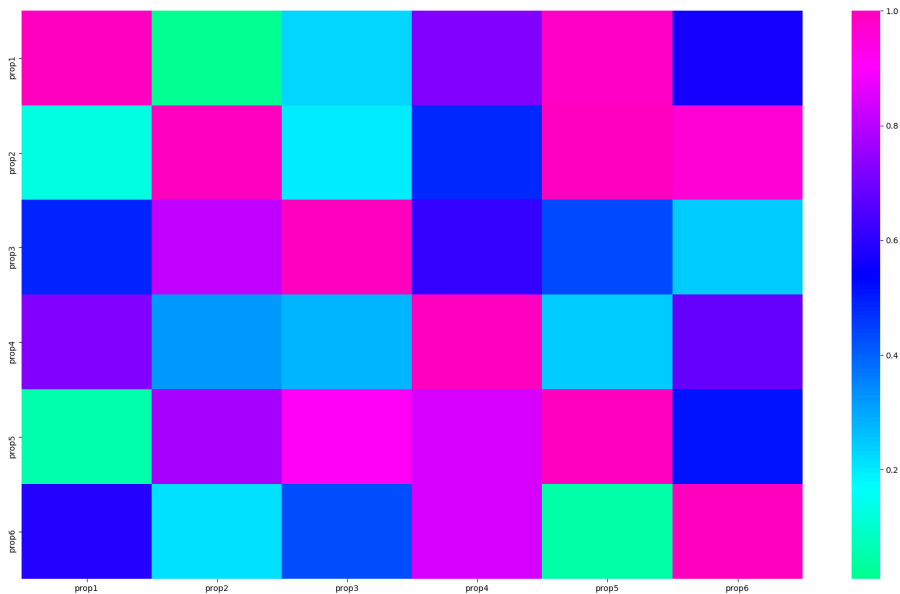
```

# 导入库
import seaborn as sns
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# 生成数据集
data = np.random.random((6,6))
np.fill_diagonal(data,np.ones(6))
features = ["prop1", "prop2", "prop3", "prop4", "prop5",
"prop6"]
data = pd.DataFrame(data, index = features,
columns=features)
print(data)

# 绘制热力图
heatmap_plot = sns.heatmap(data, center=0,
cmap='gist_rainbow')
plt.show()

```



## 24 matplotlib折线图

模块名称：example\_utils.py，里面包括三个函数，各自功能如下：

```

import matplotlib.pyplot as plt

# 创建画图fig和axes
def setup_axes():
    fig, axes = plt.subplots(ncols=3, figsize=(6.5,3))
    for ax in fig.axes:
        ax.set(xticks=[], yticks=[])

```

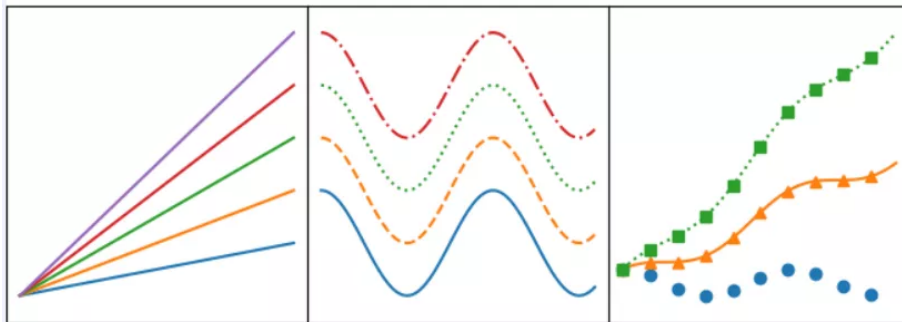
```

fig.subplots_adjust(wspace=0, left=0, right=0.93)
return fig, axes

# 图片标题
def title(fig, text, y=0.9):
    fig.suptitle(text, size=14, y=y, weight='semibold',
x=0.98, ha='right',
                bbox=dict(boxstyle='round',
fc='floralwhite', ec='#8B7E66',
                        lw=2))

# 为数据添加文本注释
def label(ax, text, y=0):
    ax.annotate(text, xy=(0.5, 0.00), xycoords='axes
fraction', ha='center',
                style='italic',
                bbox=dict(boxstyle='round',
facecolor='floralwhite',
                        ec='#8B7E66'))

```



```

import numpy as np
import matplotlib.pyplot as plt

import example_utils

x = np.linspace(0, 10, 100)

fig, axes = example_utils.setup_axes()
for ax in axes:
    ax.margins(y=0.10)

# 子图1 默认plot多条线, 颜色系统分配
for i in range(1, 6):
    axes[0].plot(x, i * x)

# 子图2 展示线的不同linestyle
for i, ls in enumerate(['-', '--', ':', '-.']):
    axes[1].plot(x, np.cos(x) + i, linestyle=ls)

# 子图3 展示线的不同linestyle和marker
for i, (ls, mk) in enumerate(zip(['-', '--', ':'], ['o',
'^', 's'])):
    axes[2].plot(x, np.cos(x) + i * x, linestyle=ls,
marker=mk, markevery=10)

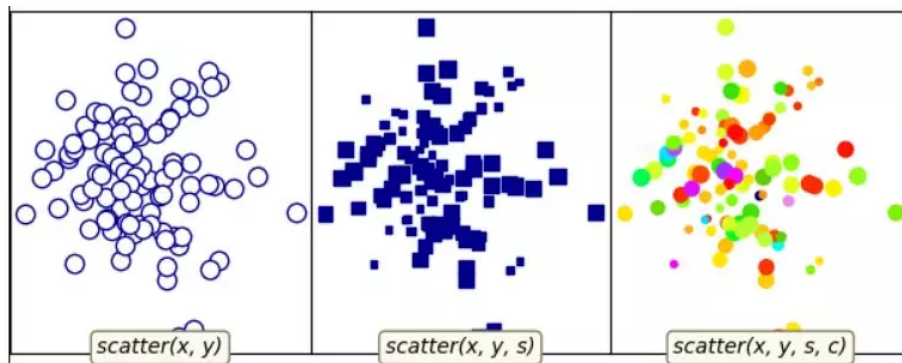
```

```

# 设置标题
# example_utils.title(fig, '"ax.plot(x, y, ...)": Lines
and/or markers', y=0.95)
# 保存图片
fig.savefig('plot_example.png', facecolor='none')
# 展示图片
plt.show()

```

## 25 matplotlib散点图



对应代码:

```

"""
散点图的基本用法
"""

import numpy as np
import matplotlib.pyplot as plt

import example_utils

# 随机生成数据
np.random.seed(1874)
x, y, z = np.random.normal(0, 1, (3, 100))
t = np.arctan2(y, x)
size = 50 * np.cos(2 * t)**2 + 10

fig, axes = example_utils.setup_axes()

# 子图1
axes[0].scatter(x, y, marker='o', color='darkblue',
facecolor='white', s=80)
example_utils.label(axes[0], 'scatter(x, y)')

# 子图2
axes[1].scatter(x, y, marker='s', color='darkblue',
s=size)
example_utils.label(axes[1], 'scatter(x, y, s)')

```

```

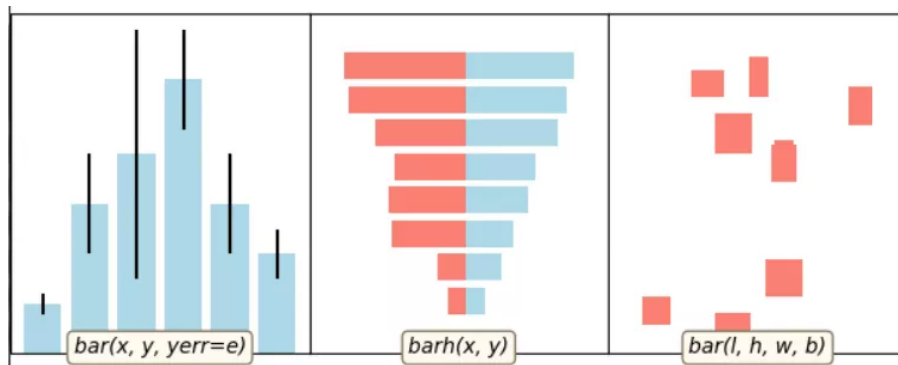
# 子图3
axes[2].scatter(x, y, s=size, c=z, cmap='gist_ncar')
example_utils.label(axes[2], 'scatter(x, y, s, c)')

# example_utils.title(fig, '"ax.scatter(...)":
Colored/scaled markers',
#                               y=0.95)
fig.savefig('scatter_example.png', facecolor='none')

plt.show()

```

## 26 matplotlib柱状图



对应代码:

```

import numpy as np
import matplotlib.pyplot as plt

import example_utils

def main():
    fig, axes = example_utils.setup_axes()

    basic_bar(axes[0])
    tornado(axes[1])
    general(axes[2])

    # example_utils.title(fig, '"ax.bar(...)": Plot
    rectangles')
    fig.savefig('bar_example.png', facecolor='none')
    plt.show()

# 子图1
def basic_bar(ax):
    y = [1, 3, 4, 5.5, 3, 2]
    err = [0.2, 1, 2.5, 1, 1, 0.5]

```

```

x = np.arange(len(y))
ax.bar(x, y, yerr=err, color='lightblue',
ecolor='black')
ax.margins(0.05)
ax.set_ylim(bottom=0)
example_utils.label(ax, 'bar(x, y, yerr=e)')

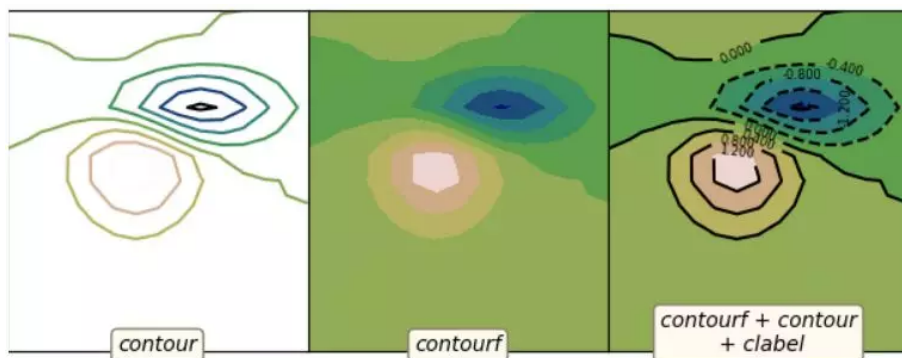
# 子图2
def tornado(ax):
    y = np.arange(8)
    x1 = y + np.random.random(8) + 1
    x2 = y + 3 * np.random.random(8) + 1
    ax.barh(y, x1, color='lightblue')
    ax.barh(y, -x2, color='salmon')
    ax.margins(0.15)
    example_utils.label(ax, 'barh(x, y)')

# 子图3
def general(ax):
    num = 10
    left = np.random.randint(0, 10, num)
    bottom = np.random.randint(0, 10, num)
    width = np.random.random(num) + 0.5
    height = np.random.random(num) + 0.5
    ax.bar(left, height, width, bottom, color='salmon')
    ax.margins(0.15)
    example_utils.label(ax, 'bar(l, h, w, b)')

main()

```

## 27 matplotlib等高线图



对应代码:

```

import matplotlib.pyplot as plt
import numpy as np
from matplotlib.cbook import get_sample_data

import example_utils

```

```

z = np.load(get_sample_data('bivariate_normal.npy'))

fig, axes = example_utils.setup_axes()

axes[0].contour(z, cmap='gist_earth')
example_utils.label(axes[0], 'contour')

axes[1].contourf(z, cmap='gist_earth')
example_utils.label(axes[1], 'contourf')

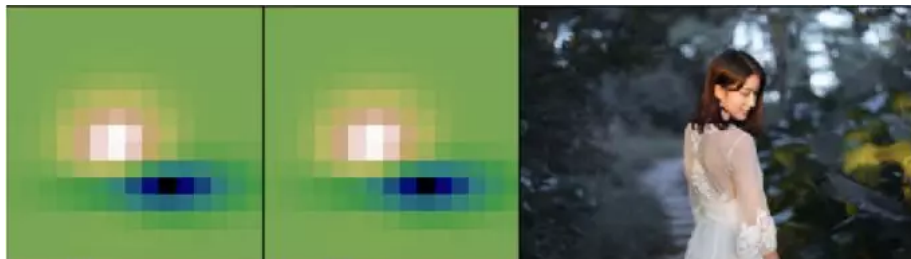
axes[2].contourf(z, cmap='gist_earth')
cont = axes[2].contour(z, colors='black')
axes[2].clabel(cont, fontsize=6)
example_utils.label(axes[2], 'contourf + contour\n +
clabel')

# example_utils.title(fig, '"contour, contourf, clabel":
Contour/label 2D data',
#                               y=0.96)
fig.savefig('contour_example.png', facecolor='none')

plt.show()

```

## 28 imshow图



对应代码:

```

import matplotlib.pyplot as plt
import numpy as np
from matplotlib.cbook import get_sample_data
from mpl_toolkits import axes_grid1

import example_utils

def main():
    fig, axes = setup_axes()
    plot(axes, *load_data())
    # example_utils.title(fig, '"ax.imshow(data, ...)":
    Colormapped or RGB arrays')
    fig.savefig('imshow_example.png', facecolor='none')
    plt.show()

```

```

def plot(axes, img_data, scalar_data, ny):

    # 默认线性插值
    axes[0].imshow(scalar_data, cmap='gist_earth', extent=
[0, ny, ny, 0])

    # 最近邻插值
    axes[1].imshow(scalar_data, cmap='gist_earth',
interpolation='nearest',
                    extent=[0, ny, ny, 0])

    # 展示RGB/RGBA数据
    axes[2].imshow(img_data)

def load_data():
    img_data = plt.imread(get_sample_data('5.png'))
    ny, nx, nbands = img_data.shape
    scalar_data =
np.load(get_sample_data('bivariate_normal.npy'))
    return img_data, scalar_data, ny

def setup_axes():
    fig = plt.figure(figsize=(6, 3))
    axes = axes_grid1.ImageGrid(fig, [0, 0, .93, 1], (1,
3), axes_pad=0)

    for ax in axes:
        ax.set(xticks=[], yticks=[])
    return fig, axes

main()

```