1. Python知识点

1.1. Python基础

1.1.1. 字符串

- 字符串表示

```
a1 = 'string1'
a2 = "string2"
'''
多行注释字符串
'''
print(a1[0])
## 字符串可像列表一样取值，切片

a1[0] = r
## 字符串不可变对象，抛出异常
```

- 字符串常见操作

方法	描述
string.capitalize()	把字符串的第一个字符大写
string.center(width)	返回一个原字符串居中,并使用空格填充至长度 width 的新字符串
string.count(str, beg=0, end=len(string))	返回 str 在 string 里面出现的次数，如果 beg 或者 end 指定则返回指定范围内 str 出现的次数
string.decode(encoding='UTF-8', errors='strict')	以 encoding 指定的编码格式解码 string，如果出错默认报一个 ValueError 的 异 常 ， 除非 errors 指 定 的是 'ignore' 或 者'replace'
string.encode(encoding='UTF-8', errors='strict')	以 encoding 指定的编码格式编码 string，如果出错默认报一个ValueError 的异常，除非 errors 指定的是'ignore'或者'replace'
string.endswith(obj, beg=0, end=len(string))	检查字符串是否以 obj 结束，如果beg 或者 end 指定则检查指定的范围内是否以 obj 结束，如果是，返回 True,否则返回 False.
string.expandtabs(tabsize=8)	把字符串 string 中的 tab 符号转为空格，tab 符号默认的空格数是 8。
string.find(str, beg=0, end=len(string))	检测 str 是否包含在 string 中，如果 beg 和 end 指定范围，则检查是否包含在指定范围内，如果是返回开始的索引值，否则返回-1
string.format()	格式化字符串
string.index(str, beg=0, end=len(string))	跟find()方法一样，只不过如果str不在 string中会报一个异常.

string.isalnum()	如果 string 至少有一个字符并且所有字符都是字母或数字则返回 True,否则返回 False
string.isalpha()	如果 string 至少有一个字符并且所有字符都是字母则返回 True,否则返回 False
string.isdecimal()	如果 string 只包含十进制数字则返回 True 否则返回 False.
string.isdigit()	如果 string 只包含数字则返回 True 否则返回 False.
string.islower()	如果 string 中包含至少一个区分大小写的字符，并且所有这些(区分大小写的)字符都是小写，则返回 True，否则返回 False
string.isnumeric()	如果 string 中只包含数字字符，则返回 True，否则返回 False
string.isspace()	如果 string 中只包含空格，则返回 True，否则返回 False.
string.istitle()	如果 string 是标题化的(见 title())则返回 True，否则返回 False
string.isupper()	如果 string 中包含至少一个区分大小写的字符，并且所有这些(区分大小写的)字符都是大写，则返回 True，否则返回 False
string.join(seq)	以 string 作为分隔符，将 seq 中所有的元素(的字符串表示)合并为一个新的字符串
string.ljust(width)	返回一个原字符串左对齐,并使用空格填充至长度 width 的新字符串
string.lower()	转换 string 中所有大写字符为小写.
string.lstrip()	截掉 string 左边的空格

string.rjust(width)	返回一个原字符串右对齐,并使用空格填充至长度 width 的新字符串
string.rpartition(str)	类似于 partition()函数,不过是从右边开始查找
string.rstrip()	删除 string 字符串末尾的空格.
string.split(str="", num=string.count(str))	以 str 为分隔符切片 string，如果 num 有指定值，则仅分隔 num+ 个子字符串
string.splitlines([keepends])	按照行('\r', '\r\n', \n')分隔，返回一个包含各行作为元素的列表，如果参数 keepends 为 False，不包含换行符，如果为 True，则保留换行符。
string.startswith(obj, beg=0, end=len(string))	检查字符串是否是以 obj 开头，是则返回 True，否则返回 False。如果beg 和 end 指定值，则在指定范围内检查.
string.strip([obj])	在 string 上执行 lstrip()和 rstrip()
string.swapcase()	翻转 string 中的大小写
string.title()	返回"标题化"的 string,就是说所有单词都是以大写开始，其余字母均为小写(见 istitle())
string.translate(str, del="")	根据 str 给出的表(包含 256 个字符)转换 string 的字符, 要过滤掉的字符放到 del 参数中
string.upper()	转换 string 中的小写字母为大写
string.zfill(width)	返回长度为 width 的字符串，原字符串 string 右对齐，前面填充0

1.1.2. List

- 有序列表，元素类型不固定，长度不固定，都可以进行的操作包括索引，切片，加，乘，检查成员

```
lst = [1,2,3,4,5]
dir(lst)
```

```
    __subclasshook__ ',
    'append',
    'clear',
    'copy',
    'count',
    'extend',
    'index',
    'insert',
    'pop',
    'remove',
    'reverse',
    'sort']
```

- 注意append方法和extend方法的区别

- sort方法的使用

```
lst = ['aa','bb','cc','dd','ee']
lst.sort(key=lambda x:x[0])
```

- List对象的内置方法

序号	方法
1	list.append(obj) 在列表末尾添加新的对象
2	list.count(obj) 统计某个元素在列表中出现的次数
3	list.extend(seq) 在列表末尾一次性追加另一个序列中的多个值（用新列表扩展原来的列表）
4	list.index(obj) 从列表中找出某个值第一个匹配项的索引位置
5	list.insert(index, obj) 将对象插入列表
6	list.pop([index=-1]) 移除列表中的一个元素（默认最后一个元素），并且返回该元素的值
7	list.remove(obj) 移除列表中某个值的第一个匹配项
8	list.reverse() 反向列表中元素
9	list.sort(cmp=None, key=None, reverse=False) 对原列表进行排序

1.1.3. 元祖(Tuple)

- 元祖不能修改元祖元素，例如：

```
ttuple = (1,2,3)
ttuple[2] = 3#抛出异常
```

- 元祖其他用法和列表类似

1.1.4. Dict(字典类)

1.1.5. 函数与函数式编程

- 函数参数：
  Python支持函数参数的写法包括：顺序传入参数，含有默认值的参数，不定长元祖参数（*args），字典参数(*kwargs)
- ◦ 不定长元祖参数详解

```
def fun_var_args(farg, *args):
    print("arg:", farg)
    for value in args:
        print("another arg:", value)
```

```
: fun_var_args(1,1,2)
```

```
arg: 1
another arg: 1
another arg: 2
```

```
: fun_var_args(1,(1,2))
```

```
arg: 1
another arg: (1, 2)
```

```
: fun_var_args(1,*(1,2))
```

```
arg: 1
another arg: 1
another arg: 2
```

注意三种传参用法的区别，第二种将tuple(1,2)当做一个参数，而第三种将tuple当做一个参数列表传入

- ◦ 字典参数详解

```
def fun_var_kwargs(farg, **kwargs):
    print ("arg:", farg)
    for key in kwargs:
        print ("another keyword arg: %s: %s" % (key, kwargs[key]))
fun_var_kwargs(1,**{"1":"2","2":"3"})
#same as fun_var_kwargs(farg=1,**{"1":"2","2":"3"})
#result:
#another keyword arg: 1: 2
#another keyword arg: 2: 3
```

- 函数式编程
- - 传递函数及函数作参数

    Python中函数可以作为值进行传递，可以理解为C语言中的函数指针，在面向对象的编程思想中，可以理解为函数为第一类对象，每一个函数就是一个类，且拥有默认的init函数和call函数

```
def foo():
    print('hello')
boo = foo
boo()# hello
def foo1(func1):
    func1()
foo1(foo)#hello
```

- - 匿名函数lambda

    python 允许用 lambda 关键字创造匿名函数。匿名是因为不需要以标准的方式来声明，比如说，使用 def 语句（除非赋值给一个局部变量，这样的对象也不会在任何的名称空间内创建名字）。然而，作为函数，它们也能有参数。一个完整的 lambda “语句”代表了一个表达式，这个表达式的定义体必须和声明放在同一行。我们现在来演示下匿名函数的语法:

- - Python函数式编程

    Python中支持三种内建函数，map(),filter(),reduce(),均可作用于可迭代对象（什么是可迭代对象后文解释），具体用法如下:

```
from functools import reduce
lst = [1,2,3,5,4,6,2,9]
# map
print(list(map(lambda x:x+1,lst)))
#filter
print(list(filter(lambda x:x%2==0,lst)))
#reduce
print(reduce(lambda x,y:x+y,lst))
#result1:[2, 3, 4, 6, 5, 7, 3, 10]
#result2:[2, 4, 6, 2]
#result3:32
```

1.1.6. Python模块

- 定义

    模块支持从逻辑上组织 Python 代码。当代码量变得相当大的时候，我们最好把代码分成一些有组织的代码段，前提是保证它们的彼此交互。这些代码片段相互间有一定的联系，可能是一个包含数据成员和方法的类，也可能是一组相关但彼此独立的操作函数。这些代码段是共享的，所以 Python 允许“调入”一个模块，允许使用其他模块的属性来利用之前的工作成果，实现代码重用。这个把其他模块中属性附加到你的模块中的操作叫做导入（import）。那些自我包含并且有组织的代码片段就是模块（module）。

在功能逻辑上上一组定义的Python函数及Python类可以看作为一个模块，在文件上，一个.py文件可以看成一个Python模块，都可以通过import方法导入。

- 导入

```
import numpy
import pandas as pd
import numpy as np,pandas as pd
from sklearn.linear_model import LinearRegression as lr
from sklearn.linear_model import LinearRegression as lr,bayes as by ,LogisticRegression as lgr
```

- 导入和加载
  模块被导入是，模块就会被加载，同时模块的顶层代码会被执行（没有包含于任何类或函数的代码），包括设定一些模块中的全局变量等。
  一个模块只会被加载一次无论它被import多少次，例如模块a.py和模块b.py都import了sys模块，但是a.py导入时加载了sys模块，b.py导入时不会再加载sys
- **尽量避免使用**from module import *
- Python包
  为了避免模块名冲突，Python又引入了按目录来组织模块的方法，称为包（Package）。
  举个例子，一个abc.py的文件就是一个名字叫abc的模块，一个xyz.py的文件就是一个名字叫xyz的模块。
  现在，假设我们的abc和xyz这两个模块名字与其他模块冲突了，于是我们可以通过包来组织模块，避免冲突。方法是选择一个顶层包名，比如mycompany，按照如下目录存放：
  引入了包以后，只要顶层的包名不与别人冲突，那所有模块都不会与别人冲突。现在，abc.py模块的名字就变成了mycompany.abc，类似的，xyz.py的模块名变成了mycompany.xyz。
  每一个包目录下面都会有一个__init__.py的文件，这个文件是必须存在的，否则，Python就把这个目录当成普通目录，而不是一个包。**init**.py可以是空文件，也可以有Python代码，因为__init__.py本身就是一个模块，而它的模块名就是mycompany。
  包可以有多级目录，组成多级层次的包结构。

```
mycompany
├─ __init__.py
├─ abc.py
└─ xyz.py
```

```
mycompany
 ├─ web
 │  ├─ __init__.py
 │  ├─ utils.py
 │  └─ www.py
 ├─ __init__.py
 ├─ abc.py
 └─ xyz.py
```

1.1.7. Python异常

- 异常的定义(来自CorePython)

  对异常的最好描述是：它是因为程序出现了错误而在正常控制流以外采取的行为。这个行为又分为两个阶段：首先是引起异常发生的错误，然后是检测（和采取可能的措施）阶段。

  第一个阶段是在发生了一个异常条件（有时候也叫做例外的条件）后发生的。只要检测到错误并且意识到异常条件，解释器会引发一个异常。引发也可以叫做触发，抛出或者生成。解释器通过它通知当前控制流有错误发生。Python 也允许程序员自己引发异常。无论是 Python 解释器还是程序员引发的，异常就是错误发生的信号。当前流将被打断，用来处理这个错误并采取相应的操作。这就是第二阶段。

  对异常的处理发生在第二阶段，异常引发后，可以调用很多不同的操作。可以是忽略错误（记录错误但不采取任何措施，采取补救措施后终止程序），或是减轻问题的影响后设法继续执行程序。所有的这些操作都代表一种继续，或是控制的分支。关键是程序员在错误发生时可以指示程序如何执行。

  你可能已经得出这样一个结论：程序运行时发生的错误主要是由于外部原因引起的，例如非法输入或是其他操作失败等。这些因素并不在程序员的直接控制下，而程序员只能预见一部分错误，编写常见的补救措施代码。

  主要目的是为了在开发人员推测可能发生错误的时候采取相应的补救措施。

- Python异常处理的方法

```
try:
     <语句>
except <异常名1>:
      print('异常说明1')
except <异常名2>:
      print('异常说明2')
except <异常名3>:
      traceback.print_exc(file=open('tb.txt','w+'))#打印完整异常栈
else:
      doSomething()#未捕获异常执行
finally:
      pass#无论如何最后都会执行
```

- 自定义异常

```
#继承异常类
class MyError(Exception):
    def __init__(self, value):
        self.value = value
    def __str__(self):
        return repr(self.value)

try:
    raise MyError(2*2)
except MyError as e:
    print('My exception occurred, value:', e.value)
```

- 断言语法Assert
  assert断言等价于

```
if express not true:
    raise AssertionError
```

常见的写法包括

```
assert True     # 条件为 true 正常执行
assert False    # 条件为 false 触发异常
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AssertionError
assert 1==1    # 条件为 true 正常执行
assert 1==2    # 条件为 false 触发异常
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AssertionError

assert 1==2, '1 不等于 2'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AssertionError: 1 不等于 2
```

1.1.8. Python面向对象编程

1.1.9. Python列表推导式与生成式

1.1.10. Python装饰器