14. 前方高能-内置函数二

本节主要内容:

1. lamda匿名函数
2. sorted()
3. filter()
4. map()
5. 递归函数

一. lamda匿名函数

为了解决一些简单的需求而设计的一句话函数

```
# 计算n的n次方
def func(n):
    return n**n
print(func(10))


f = lambda n: n**n
print(f(10))
```

lambda表示的是匿名函数. 不需要用def来声明, 一句话就可以声明出一个函数

语法:

函数名 = lambda 参数: 返回值

注意:

1. 函数的参数可以有多个. 多个参数之间用逗号隔开
2. 匿名函数不管多复杂. 只能写一行, 且逻辑结束后直接返回数据
3. 返回值和正常的函数一样, 可以是任意数据类型

匿名函数并不是说一定没有名字. 这里前面的变量就是一个函数名. 说他是匿名原因是我们通过__name__查看的时候是没有名字的. 统一都叫lambda. 在调用的时候没有什么特别之处. 像正常的函数调用即可

二. sorted()

排序函数.

语法: sorted(Iterable, key=None, reverse=False)

Iterable: 可迭代对象

key: 排序规则(排序函数), 在sorted内部会将可迭代对象中的每一个元素传递给这个函数的参数. 根据函数运算的结果进行排序

reverse: 是否是倒叙. True: 倒叙, False: 正序

```
lst = [1,5,3,4,6]
lst2 = sorted(lst)
print(lst)  # 原列表不会改变
print(lst2) # 返回的新列表是经过排序的

dic = {1:'A', 3:'C', 2:'B'}
print(sorted(dic))  # 如果是字典. 则返回排序过后的key
```

和函数组合使用

```
# 根据字符串长度进行排序
lst = ["麻花藤", "冈本次郎", "中央情报局", "狐仙"]

# 计算字符串长度
def func(s):
    return len(s)

print(sorted(lst, key=func))
```

和lambda组合使用

```
# 根据字符串长度进行排序
lst = ["麻花藤", "冈本次郎", "中央情报局", "狐仙"]

# 计算字符串长度
def func(s):
    return len(s)

print(sorted(lst, key=lambda s: len(s)))


lst = [{"id":1, "name":'alex', "age":18},
       {"id":2, "name":'wusir', "age":16},
       {"id":3, "name":'taibai', "age":17}]
# 按照年龄对学生信息进行排序
print(sorted(lst, key=lambda e: e['age']))
```

三. filter()

筛选函数

语法: filter(function. Iterable)

function: 用来筛选的函数. 在filter中会自动的把iterable中的元素传递给function. 然后根据function返回的True或者False来判断是否保留此项数据

Iterable: 可迭代对象

```
lst = [1,2,3,4,5,6,7]
ll = filter(lambda x: x%2==0, lst)    # 筛选所有的偶数
print(ll)
```

```
print(list(ll))

lst = [{"id":1, "name":'alex', "age":18},
       {"id":2, "name":'wusir', "age":16},
       {"id":3, "name":'taibai', "age":17}]

fl = filter(lambda e: e['age'] > 16, lst)   # 筛选年龄大于16的数据
print(list(fl))
```

四. map()

映射函数

语法: map(function, iterable) 可以对可迭代对象中的每一个元素进行映射. 分别取执行function

计算列表中每个元素的平方 ,返回新列表

```
def func(e):
    return e*e

mp = map(func, [1, 2, 3, 4, 5])
print(mp)
print(list(mp))
```

改写成lambda

```
print(list(map(lambda x: x * x, [1, 2, 3, 4, 5])))
```

计算两个列表中相同位置的数据的和

```
# 计算两个列表相同位置的数据的和
lst1 = [1, 2, 3, 4, 5]
lst2 = [2, 4, 6, 8, 10]
print(list(map(lambda x, y: x+y, lst1, lst2)))
```

五. 递归

在函数中调用函数本身. 就是递归

```
def func():
    print("我是谁")
    func()
func()
```

在python中递归的深度最大到998

```
def foo(n):
    print(n)
    n += 1
    foo(n)
foo(1)
```

递归的应用:

我们可以使用递归来遍历各种树形结构, 比如我们的文件夹系统. 可以使用递归来遍历该

文件夹中的所有文件

```
import os

def read(filepath, n):
    files = os.listdir(filepath)     # 获取到当前文件夹中的所有文件
    for fi in files:     # 遍历文件夹中的文件，这里获取的只是本层文件名
        fi_d = os.path.join(filepath,fi)     # 加入文件夹 获取到文件夹+文件
        if os.path.isdir(fi_d): # 如果该路径下的文件是文件夹
            print("\t"*n, fi)
            read(fi_d, n+1)      # 继续进行相同的操作
        else:
            print("\t"*n, fi)   # 递归出口. 最终在这里隐含着return


#递归遍历目录下所有文件
read('../oldboy/', 0)
```

六. 二分查找

二分查找. 每次能够排除掉一半的数据. 查找的效率非常高. 但是局限性比较大. 必须是有序序列才可以使用二分查找

要求: 查找的序列必须是有序序列.

```
# 判断n是否在lst中出现. 如果出现请返回n所在的位置
# 二分查找---非递归算法
lst = [22, 33, 44, 55, 66, 77, 88, 99, 101, 238, 345, 456, 567, 678, 789]
n = 567
left = 0
right = len(lst) - 1
count = 1
while left <= right:
    middle = (left + right) // 2
    if n < lst[middle]:
        right = middle - 1
    elif n > lst[middle]:
        left = middle + 1
    else:
        print(count)
        print(middle)
        break
    count = count + 1
else:
    print("不存在")


# 普通递归版本二分法
def binary_search(n, left, right):
    if left <= right:
        middle = (left+right) // 2
        if n < lst[middle]:
```

```
            right = middle - 1
        elif n > lst[middle]:
            left = middle + 1
        else:
            return middle
        return binary_search(n, left, right)    # 这个return必须要加. 否则接收到的永远是None.

    else:
        return -1

print(binary_search(567, 0, len(lst)-1))


# 另类二分法, 很难计算位置.
def binary_search(ls, target):

    left = 0
    right = len(ls) - 1
    if left > right:
        print("不在这里")
    middle = (left + right) // 2
    if target < ls[middle]:
        return binary_search(ls[:middle], target)
    elif target > ls[middle]:
        return binary_search(ls[middle+1:], target)
    else:
        print("在这里")

binary_search(lst, 567)
```