课程：函数加强

目标

- 应用：学员管理系统
- 递归
- lambda 表达式
- 高阶函数

一. 应用：学员管理系统

1.1 系统简介

需求：进入系统显示系统功能界面，功能如下：

- 1、添加学员
- 2、删除学员
- 3、修改学员信息
- 4、查询学员信息
- 5、显示所有学员信息
- 6、退出系统

系统共6个功能，用户根据自己需求选取。

1.2 步骤分析

1. 显示功能界面
2. 用户输入功能序号
3. 根据用户输入的功能序号，执行不同的功能(函数)

   3.1 定义函数

   3.2 调用函数

1.3 需求实现

1.3.1 显示功能界面

定义函数`print_info`，负责显示系统功能。

```python
def print_info():
    print('-' * 20)
    print('欢迎登录学员管理系统')
    print('1: 添加学员')
    print('2: 删除学员')
    print('3: 修改学员信息')
    print('4: 查询学员信息')
    print('5: 显示所有学员信息')
    print('6: 退出系统')
    print('-' * 20)


print_info()
```

1.3.2 用户输入序号，选择功能

```python
user_num = input('请选择您需要的功能序号: ')
```

1.3.3 根据用户选择，执行不同的功能

```python
if user_num == '1':
    print('添加学员')
elif user_num == '2':
    print('删除学员')
elif user_num == '3':
    print('修改学员信息')
elif user_num == '4':
    print('查询学员信息')
elif user_num == '5':
    print('显示所有学员信息')
elif user_num == '6':
    print('退出系统')
```

> 工作中，需要根据实际需求调优代码。
>
> 1. 用户选择系统功能的代码需要循环使用，直到用户主动退出系统。
> 2. 如果用户输入1-6以外的数字，需要提示用户。

```python
while True:
    # 1. 显示功能界面
    print_info()

    # 2. 用户选择功能
    user_num = input('请选择您需要的功能序号: ')

    # 3. 根据用户选择，执行不同的功能
    if user_num == '1':
```

```
        print('添加学员')
    elif user_num == '2':
        print('删除学员')
    elif user_num == '3':
        print('修改学员信息')
    elif user_num == '4':
        print('查询学员信息')
    elif user_num == '5':
        print('显示所有学员信息')
    elif user_num == '6':
        print('退出系统')
    else:
        print('输入错误，请重新输入!!!')
```

1.3.4 定义不同功能的函数

所有功能函数都是操作学员信息，所有存储所有学员信息应该是一个全局变量，数据类型为列表。

```
info = []
```

1.3.4.1 添加学员

- 需求分析

1. 接收用户输入学员信息，并保存
2. 判断是否添加学员信息

   2.1 如果学员姓名已经存在，则报错提示

   2.2 如果学员姓名不存在，则准备空字典，将用户输入的数据追加到字典，再列表追加字典数据
3. 对应的if条件成立的位置调用该函数

- 代码实现

```
def add_info():
    """ 添加学员 """
    # 接收用户输入学员信息
    new_id = input('请输入学号: ')
    new_name = input('请输入姓名: ')
    new_tel = input('请输入手机号: ')


    # 声明info是全局变量
    global info

    # 检测用户输入的姓名是否存在，存在则报错提示
    for i in info:
        if new_name == i['name']:
```

```
            print('该用户已经存在！')
            return

    # 如果用户输入的姓名不存在，则添加该学员信息
    info_dict = {}

    # 将用户输入的数据追加到字典
    info_dict['id'] = new_id
    info_dict['name'] = new_name
    info_dict['tel'] = new_tel

    # 将这个学员的字典数据追加到列表
    info.append(info_dict)

    print(info)
```

1.3.4.2 删除学员

- 需求分析

按用户输入的学员姓名进行删除

1. 用户输入目标学员姓名
2. 检查这个学员是否存在

   2.1 如果存在，则列表删除这个数据

   2.2 如果不存在，则提示“该用户不存在”
3. 对应的if条件成立的位置调用该函数

- 代码实现

```
# 删除学员
def del_info():
    """删除学员"""
    # 1. 用户输入要删除的学员的姓名
    del_name = input('请输入要删除的学员的姓名：')

    global info
    # 2. 判断学员是否存在:如果输入的姓名存在则删除，否则报错提示
    for i in info:
        if del_name == i['name']:
            info.remove(i)
            break
    else:
        print('该学员不存在')

    print(info)
```

1.3.4.3 修改学员信息

- 需求分析

1. 用户输入目标学员姓名
2. 检查这个学员是否存在

   2.1 如果存在，则修改这位学员的信息，例如手机号

   2.2 如果不存在，则报错
3. 对应的if条件成立的位置调用该函数

- 代码实现

```
# 修改函数
def modify_info():
    """修改函数"""
    # 1. 用户输入要修改的学员的姓名
    modify_name = input('请输入要修改的学员的姓名：')

    global info
    # 2. 判断学员是否存在：如果输入的姓名存在则修改手机号，否则报错提示
    for i in info:
        if modify_name == i ['name']:
            i['tel'] = input('请输入新的手机号：')
            break
    else:
        print('该学员不存在')

    print(info)
```

1.3.4.4 查询学员信息

- 需求分析

1. 用户输入目标学员姓名
2. 检查学员是否存在

   2.1 如果存在，则显示这个学员的信息

   2.2 如果不存在，则报错提示
3. 对应的if条件成立的位置调用该函数

- 代码实现

```
# 查询学员
def search_info():
    """查询学员"""
    # 1. 输入要查找的学员姓名：
    search_name = input('请输入要查找的学员姓名：')

```

```
    global info
    # 2. 判断学员是否存在：如果输入的姓名存在则显示这位学员信息，否则报错提示
    for i in info:
        if search_name == i['name']:
            print('查找到的学员信息如下：----------')
            print(f"该学员的学号是{i['id']}，姓名是{i['name']}，手机号是
{i['tel']}")
            break
    else:
        print('该学员不存在')
```

1.3.4.5 显示所有学员信息

- 需求分析

打印所有学员信息

- 代码实现

```
# 显示所有学员信息
def print_all():
    """ 显示所有学员信息 """
    print('学号\t姓名\t手机号')
    for i in info:
        print(f'{i["id"]}\t{i["name"]}\t{i["tel"]}')
```

1.3.4.6 退出系统

在用户输入功能序号`6`的时候要退出系统，代码如下：

```
    ......
    elif user_num == '6':
        exit_flag = input('确定要退出吗？yes or no')
        if exit_flag == 'yes':
            break
```

二. 递归

2.1 递归的应用场景

递归是一种编程思想，应用场景：

1. 在我们日常开发中，如果要遍历一个文件夹下面所有的文件，通常会使用递归来实现；
2. 在后续的算法课程中，很多算法都离不开递归，例如：快速排序。

2.1.1 递归的特点

- 函数内部自己调用自己
- 必须有出口

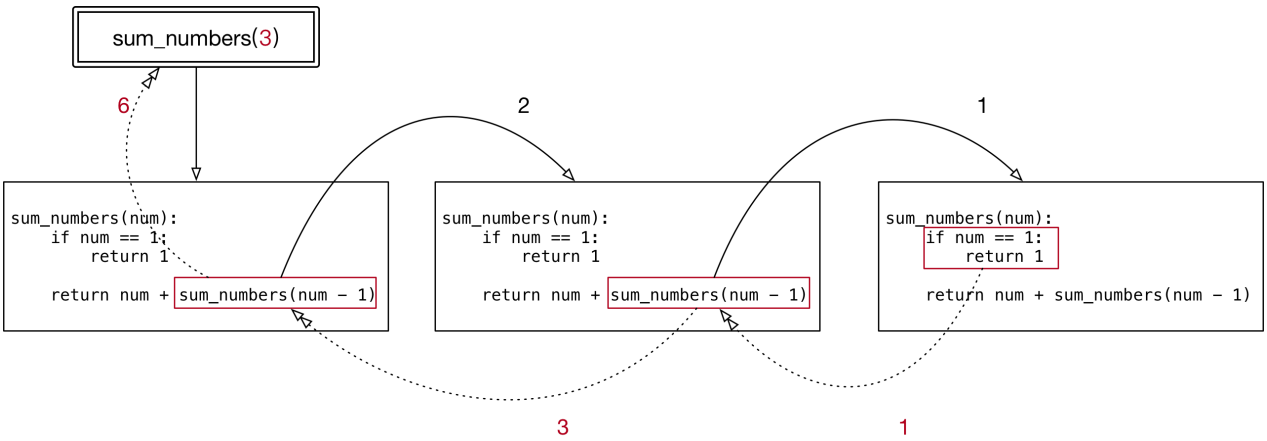
2.2 应用：3以内数字累加和

- 代码

```python
# 3 + 2 + 1
def sum_numbers(num):
    # 1.如果是1，直接返回1 -- 出口
    if num == 1:
        return 1
    # 2.如果不是1，重复执行累加并返回结果
    return num + sum_numbers(num-1)


sum_result = sum_numbers(3)
# 输出结果为6
print(sum_result)
```

- 执行结果



三. lambda 表达式

3.1 lambda的应用场景

如果一个函数有一个返回值，并且只有一句代码，可以使用 lambda简化。

3.2 lambda语法

```
lambda 参数列表 ： 表达式
```

> 注意

- lambda表达式的参数可有可无，函数的参数在lambda表达式中完全适用。
- lambda表达式能接收任何数量的参数但只能返回一个表达式的值。

快速入门

```python
# 函数
def fn1():
    return 200


print(fn1)
print(fn1())


# lambda表达式
fn2 = lambda: 100
print(fn2)
print(fn2())
```

> 注意：直接打印lambda表达式，输出的是此lambda的内存地址

3.3 示例：计算a + b

3.3.1 函数实现

```python
def add(a, b):
    return a + b


result = add(1, 2)
print(result)
```

> 思考：需求简单，是否代码多?

3.3.2 lambda实现

```python
fn1 = lambda a, b: a + b
print(fn1(1, 2))
```

3.4 lambda的参数形式

3.4.1.无参数

```
fn1 = lambda: 100
print(fn1())
```

3.4.2.一个参数

```
fn1 = lambda a: a
print(fn1('hello world'))
```

3.4.3.默认参数

```
fn1 = lambda a, b, c=100: a + b + c
print(fn1(10, 20))
```

3.4.4.可变参数：*args

```
fn1 = lambda *args: args
print(fn1(10, 20, 30))
```

> 注意：这里的可变参数传入到lambda之后，返回值为元组。

3.4.5.可变参数：**kwargs

```
fn1 = lambda **kwargs: kwargs
print(fn1(name='python', age=20))
```

3.5 lambda的应用

3.5.1. 带判断的lambda

```
fn1 = lambda a, b: a if a > b else b
print(fn1(1000, 500))
```

3.5.2. 列表数据按字典key的值排序

```
students = [
    {'name': 'TOM', 'age': 20},
    {'name': 'ROSE', 'age': 19},
```

```
    {'name': 'Jack', 'age': 22}
]

# 按name值升序排列
students.sort(key=lambda x: x['name'])
print(students)

# 按name值降序排列
students.sort(key=lambda x: x['name'], reverse=True)
print(students)

# 按age值升序排列
students.sort(key=lambda x: x['age'])
print(students)
```

四. 高阶函数

把函数作为参数传入，这样的函数称为高阶函数，高阶函数是函数式编程的体现。函数式编程就是指这种高度抽象的编程范式。

4.1 体验高阶函数

在Python中，`abs()`函数可以完成对数字求绝对值计算。

```
abs(-10)  # 10
```

`round()`函数可以完成对数字的四舍五入计算。

```
round(1.2)  # 1
round(1.9)  # 2
```

需求：任意两个数字，按照指定要求整理数字后再进行求和计算。

- 方法1

```
def add_num(a, b):
    return abs(a) + abs(b)


result = add_num(-1, 2)
print(result)  # 3
```

- 方法2

```python
def sum_num(a, b, f):
    return f(a) + f(b)


result = sum_num(-1, 2, abs)
print(result)  # 3
```

> 注意：两种方法对比之后，发现，方法2的代码会更加简洁，函数灵活性更高。

函数式编程大量使用函数，减少了代码的重复，因此程序比较短，开发速度较快。

4.2 内置高阶函数

4.2.1 map()

map(func, lst)，将传入的函数变量func作用到lst变量的每个元素中，并将结果组成新的列表(Python2)/迭代器(Python3)返回。

需求：计算`list1`序列中各个数字的2次方。

```python
list1 = [1, 2, 3, 4, 5]


def func(x):
    return x ** 2


result = map(func, list1)

print(result)  # <map object at 0x0000013769653198>
print(list(result))  # [1, 4, 9, 16, 25]
```

4.2.2 reduce()

reduce(func，lst)，其中func必须有两个参数。每次func计算的结果继续和序列的下一个元素做累积计算。

> 注意：reduce()传入的参数func必须接收2个参数。

需求：计算`list1`序列中各个数字的累加和。

```python
import functools

list1 = [1, 2, 3, 4, 5]


def func(a, b):
    return a + b


result = functools.reduce(func, list1)

print(result)  # 15
```

4.2.3 filter()

filter(func, lst)函数用于过滤序列, 过滤掉不符合条件的元素, 返回一个 filter 对象。如果要转换为列表, 可以使用 list() 来转换。

```python
list1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]


def func(x):
    return x % 2 == 0


result = filter(func, list1)

print(result)  # <filter object at 0x0000017AF9DC3198>
print(list(result))  # [2, 4, 6, 8, 10]
```

五. 总结

- 递归
  - 函数内部自己调用自己
  - 必须有出口
- lambda
  - 语法

    ```
    lambda 参数列表: 表达式
    ```

  - lambda的参数形式
    - 无参数

```
lambda: 表达式
```

- 一个参数

```
lambda 参数: 表达式
```

- 默认参数

```
lambda key=value: 表达式
```

- 不定长位置参数

```
lambda *args: 表达式
```

- 不定长关键字参数

```
lambda **kwargs: 表达式
```

- 高阶函数
  - 作用：把函数作为参数传入，化简代码
  - 内置高阶函数
    - map()
    - reduce()
    - filter()