

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：

- 添加学员
- 删除学员
- 修改学员信息
- 查询学员信息
- 显示所有学员信息
- 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。

```

1  def print_info():
2      print('-' * 20)
3      print('欢迎登录学员管理系统')
4      print('1: 添加学员')
5      print('2: 删除学员')
6      print('3: 修改学员信息')
7      print('4: 查询学员信息')
8      print('5: 显示所有学员信息')
9      print('6: 退出系统')
10     print('-' * 20)
11
12
13  print_info()

```

1.3.2 用户输入序号，选择功能

```

1  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

```

1  if user_num == '1':
2      print('添加学员')
3  elif user_num == '2':
4      print('删除学员')
5  elif user_num == '3':
6      print('修改学员信息')
7  elif user_num == '4':
8      print('查询学员信息')
9  elif user_num == '5':
10     print('显示所有学员信息')
11  elif user_num == '6':
12     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。

```

1  while True:
2      # 1. 显示功能界面
3      print_info()
4
5      # 2. 用户选择功能
6      user_num = input('请选择您需要的功能序号: ')
7
8      # 3. 根据用户选择，执行不同的功能
9      if user_num == '1':

```

```

10     print('添加学员')
11     elif user_num == '2':
12         print('删除学员')
13     elif user_num == '3':
14         print('修改学员信息')
15     elif user_num == '4':
16         print('查询学员信息')
17     elif user_num == '5':
18         print('显示所有学员信息')
19     elif user_num == '6':
20         print('退出系统')
21     else:
22         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。

```

1 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

```

1 def add_info():
2     """ 添加学员 """
3     # 接收用户输入学员信息
4     new_id = input('请输入学号: ')
5     new_name = input('请输入姓名: ')
6     new_tel = input('请输入手机号: ')
7
8
9     # 声明info是全局变量
10    global info
11
12    # 检测用户输入的姓名是否存在，存在则报错提示
13    for i in info:
14        if new_name == i['name']:

```

```

15         print('该用户已经存在! ')
16         return
17
18     # 如果用户输入的姓名不存在, 则添加该学员信息
19     info_dict = {}
20
21     # 将用户输入的数据追加到字典
22     info_dict['id'] = new_id
23     info_dict['name'] = new_name
24     info_dict['tel'] = new_tel
25
26     # 将这个学员的字典数据追加到列表
27     info.append(info_dict)
28
29     print(info)

```

1.3.4.2 删除学员

- 需求分析

按用户输入的学员学号进行删除

1. 用户输入目标学员学号
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, 则报错, 并重新输入
3. 对应的if条件成立的位置调用该函数

- 代码实现

```

1  def del_info():
2      """ 删除学员 """
3      while True:
4          del_id = int(input('请输入要删除的学员学号: '))
5
6          global info
7          # 检查学员是否存在
8          # 如果存在则删除列表指定下标的数据
9          if 0 <= del_id < len(info):
10             del_flag = input('确定要删除吗? yes or no')
11             if del_flag == 'yes':
12                 del info[del_id]
13                 print(info)
14                 # 删除了目标学员信息后退出循环
15                 break
16             else:
17                 print('输入学员有误, 请重新输入')

```

1.3.4.3 修改学员信息

- 需求分析
 1. 用户输入目标学员学号
 2. 检查这个学员是否存在
 - 2.1 如果存在，显示此学员信息，并供用户输入新的信息
 - 2.2 如果不存在，则报错，并重新输入
 3. 对应的if条件成立的位置调用该函数
- 代码实现

```
1 def modify_info():
2     """ 修改学员信息 """
3     while True:
4         # 用户输入要修改的学员学号
5         modify_num = int(input('请输入要修改的学员学号: '))
6
7         global info
8
9         # 检查这个学员是否存在，存在则打印学员信息，并按用户输入修改
10        if 0 <= modify_num < len(info):
11            print(f'该学员学号是{info[modify_num]["id"]}, 姓名是{info[modify_num]
12            ["name"]}, 手机号是{info[modify_num]["tel"]}')
13            info[modify_num]["id"] = input('请输入学号:')
14            info[modify_num]["name"] = input('请输入姓名:')
15            info[modify_num]["tel"] = input('请输入手机号:')
16            print(info)
17            break
18        else:
19            print('输入学员有误，请重新输入')
```

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

```

1  def search_info():
2      """ 查询学员信息 """
3      search_name = input('请输入要查找的学员姓名: ')
4
5      for i in info:
6          if search_name == i['name']:
7              print('*** 查询到信息如下 ***')
8              print(f'该学员学号是{i["id"]}, 姓名是{i["name"]}, 手机号是{i["tel"]}')
9              break
10     else:
11         print('查无此人.....')

```

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

```

1  # 显示所有学员信息
2  def print_all():
3      """ 显示所有学员信息 """
4      print('学号\t姓名\t手机号')
5      for i in info:
6          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：

```

1      .....
2      elif user_num == '6':
3          exit_flag = input('确定要退出吗? yes or no')
4          if exit_flag == 'yes':
5              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

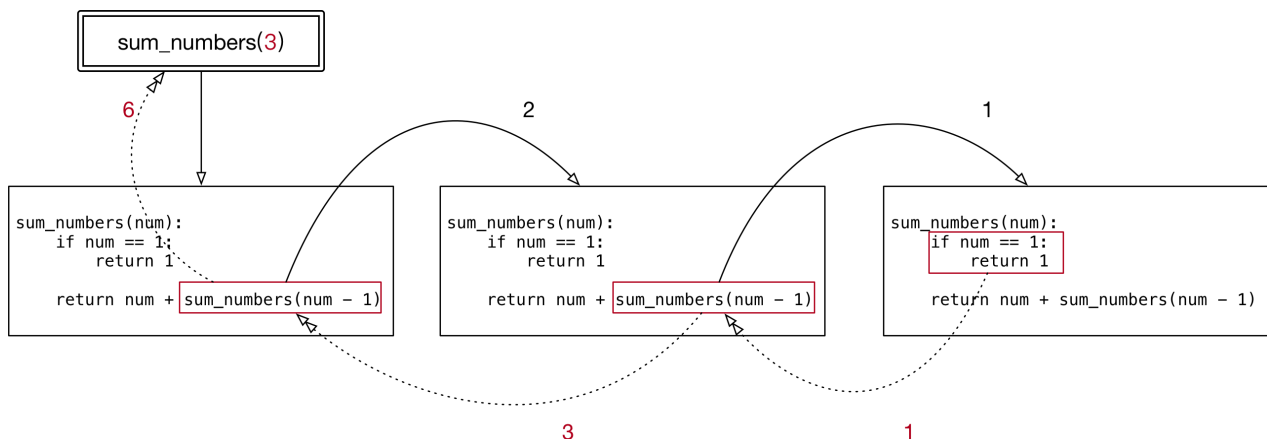
- 函数内部自己调用自己
- 必须有出口

2.2 应用：3以内数字累加和

- 代码

```
1  # 3 + 2 + 1
2  def sum_numbers(num):
3      # 1.如果是1, 直接返回1 -- 出口
4      if num == 1:
5          return 1
6      # 2.如果不是1, 重复执行累加:
7      result = num + sum_numbers(num-1)
8
9      # 3.返回累加结果
10     return result
11
12
13 sum_result = sum_numbers(3)
14 # 输出结果为6
15 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

```
1 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函数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

```
1 # 函数
2 def fn1():
3     return 200
4
5
6 print(fn1)
7 print(fn1())
8
9
10 # lambda表达式
11 fn2 = lambda: 100
12 print(fn2)
13 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

```
1 def add(a, b):
2     return a + b
3
4
5 result = add(1, 2)
6 print(result)
```

思考：需求简单，是否代码多？

3.3.2 lambda实现


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

3.4 lambda的参数形式

3.4.1.无参数

```
1 print((lambda: 100)())
```

3.4.2.一个参数

```
1 print((lambda a: a)('hello world'))
```

3.4.3.默认参数

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

3.4.4.可变参数: **args

```
1 print((lambda *args: args)(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

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

3.5 lambda的应用

3.5.1. 带判断的lambda

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

3.5.2. 列表数据按字典key的值排序

```
1 students = [  
2     {'name': 'TOM', 'age': 20},  
3     {'name': 'ROSE', 'age': 19},  
4     {'name': 'Jack', 'age': 22}
```

```

5 ]
6
7 # 按name值升序排列
8 students.sort(key=lambda x: x['name'])
9 print(students)
10
11 # 按name值降序排列
12 students.sort(key=lambda x: x['name'], reverse=True)
13 print(students)
14
15 # 按age值升序排列
16 students.sort(key=lambda x: x['age'])
17 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

需求：一个函数完成计算任意两个数字的绝对值之和。

在Python中，`abs()` 函数可以完成对数字求绝对值计算。

```

1 abs(-10) # 10

```

- 方法1

```

1 def add_num(a, b):
2     return abs(a) + abs(b)
3
4
5 result = add_num(-1, 2)
6 print(result) # 3

```

- 方法2

```

1 def sum_num(a, b, f):
2     return f(a) + f(b)
3
4
5 result = sum_num(-1, 2, abs)
6 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。

```
1 list1 = [1, 2, 3, 4, 5]
2
3
4 def func(x):
5     return x ** 2
6
7
8 result = map(func, list1)
9
10 print(result) # <map object at 0x0000013769653198>
11 print(list(result)) # [1, 4, 9, 16, 25]
```

4.2.2 reduce()

reduce(func(x,y), 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必须接受2个参数。

需求：计算 list1 序列中各个数字的累加和。

```

1  import functools
2
3  list1 = [1, 2, 3, 4, 5]
4
5
6  def func(a, b):
7      return a + b
8
9
10 result = functools.reduce(func, list1)
11
12 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。

```

1  list1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
2
3
4  def func(x):
5      return x % 2 == 0
6
7
8  result = filter(func, list1)
9
10 print(result) # <filter object at 0x0000017AF9DC3198>
11 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

```

1  lambda 参数列表: 表达式

```

- lambda的参数形式
 - 无参数

1 | `lambda`: 表达式

- 一个参数

1 | `lambda` 参数: 表达式

- 默认参数

1 | `lambda` key=value: 表达式

- 不定长位置参数

1 | `lambda` *args: 表达式

- 不定长关键字参数

1 | `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`