day04 课堂笔记

课程之前

复习和反馈

```
list1 = [1, 2, 3]
for i in list1:
    print(i)

for i in range(n):    # 循环几次,n 就是几
    pass

补充:
    列表推导式: 快速的生成列表
变量名 = [生成数据的规则 for 变量 in xxx]  # 循环每执行一次, 在列表中添加一个数据
list1 = []
for i in range(5):
    list1.append(i)


range(start, end, step)

range(1, 10)      # 1 2 3 4 5 6 7 8 9
```

```
range(1, 10, 1)  # 1 2 3 4 5 6 7 8 9
range(1, 10, 2)  # 1 3 5 7 9
range(1, 10, 3)  # 1 4 7
range(1, 10, 5)  # 1 6
```

```
>>> list1 = ['hello' for i in range(5)]
>>> list1
['hello', 'hello', 'hello', 'hello', 'hello']
>>> list2 = [i for i in range(5)]
>>> list2
[0, 1, 2, 3, 4]
>>> list3 = [i for i in list1]
>>> list3
['hello', 'hello', 'hello', 'hello', 'hello']
```

作业

```
find()  找到了返回的是下标(肯定不是-1)
没有找到  返回是-1
```

今日内容

- 列表
- 元组
- 字典

- 函数的基本概念

列表

定义

```
变量 = [数据, 数据, ...]
```

添加

```
列表.append(数据)
```

查询

```
列表.index(数据)(如果数据不存在,会报错)
列表.count(数据)
```

修改操作

```
想要修改列中的指定下标位置的数据, 使用的语法是:

列表[下标] = 数据

# 字符串中字符不能使用下标修改
```

```python
# 定义列表
my_list = [1, 3, 5, 7]

# 1. 想要将下标为 1 的数据修改为 22
my_list[1] = 22
print(my_list)  # [1, 22, 5, 7]

# 修改最后一个位置的数据，改为 'hello'
my_list[-1] = 'hello'
print(my_list)  # [1, 22, 5, 'hello']

# 2. 如果指定的下标不存在，会报错的
# my_list[10] = 10  # 代码会报错
```

删除操作

在列表中删除中间的数据，那么后面的数据会向前移动

- 根据下标删除

```
列表.pop(下标)  # 删除指定下标位置对应的数据
1. 下标不写,默认删除最后一个数据(常用)
2. 书写存在的下标，删除对应下标位置的数据

返回：返回的删除的数据
```

- 根据数据值删除

```
列表.remove(数据值)  # 根据数据值删除

返回: None
注意: 如果要删除的数据不存在, 会报错
```

- 清空数据(一般不用)

```
列表.clear()
```

```
my_list = [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]

# 1. 删除最后一个位置的数据
num = my_list.pop()
print('删除的数据为:', num)
print(my_list)  # [1, 3, 5, 7, 9, 2, 4, 6, 8]

# 2. 删除下标为 1 的数据 3
my_list.pop(1)
print(my_list)  # [1, 5, 7, 9, 2, 4, 6, 8]

# 3. 删除数据为 7 的数据
my_list.remove(7)  # 注意, 如果列表中有多个 7, 只能删除第一个, 如果数据不存在,会报错的
print(my_list)  # [1, 5, 9, 2, 4, 6, 8]
```

```
# my_list.remove(7)  # 会报错的

# 清空
my_list.clear()
print(my_list)  # []
```

列表的反转(倒置)

```
字符串中 反转倒置:  字符串[::-1]

列表中 反转和倒置:
1. 列表[::-1]  # 使用切片的方法,会得到一个新列表, 原列表不会发生改变
2. 列表.reverse()   # 直接修改原列表, 返回 None
```

```python
my_list = [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]

# 使用切片的方法反转，会得到一个新列表

list1 = my_list[::-1]
print('my_list:', my_list)
print('list1  :', list1)

# 使用 reverse 方法，直接修改原列表
my_list.reverse()
print('my_list:', my_list)
```

列表的复制

将列表中的数据复制一份,给到一个新的列表
使用场景：有一个列表，需要修改操作列表中的数据，修改之后，需要和原数据进行对比,即原数据不能改

1. 使用切片
变量 = 列表[:]

2. 使用 copy 方法
变量 = 列表.copy()

```
my_list = [1, 2, 3]

my_list1 = my_list[:]
print('my_list :', my_list)
print('my_list1:', my_list1)
my_list1[1] = 22
print('my_list :', my_list)
print('my_list1:', my_list1)
print('-' * 30)

my_list2 = my_list.copy()
print('my_list :', my_list)
print('my_list2:', my_list2)
my_list2[2] = 33
print('my_list :', my_list)
print('my_list2:', my_list2)

print('=' * 30)
my_list3 = my_list  # 这是同一个列表,多了一个名字，引用
print('my_list :', my_list)
print('my_list3:', my_list3)
my_list3[0] = 11
print('my_list :', my_list)
print('my_list3:', my_list3)
```

列表的排序

列表的排序，一般来说都是对数字进行排序的

```
列表.sort()  # 按照升序排序，从小到大

列表.sort(reverse=True)   # 降序排序，从大到小
```

```
my_list = [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]

# 升序排序
my_list.sort()
print(my_list)

# 降序排序
my_list.sort(reverse=True)

print(my_list)
```

列表嵌套

列表嵌套，列表中的内容还是列表
使用下标来确定获取的是什么类型的数据,然后确定可以继续进行什么操作

```python
person_info = [["张三", "18", "功能测试"], ["李四", "20", "自动化测试"]]

print(len(person_info))  # 2
print(person_info[0])  # ['张三', '18', '功能测试']

print(person_info[0][0])  # '张三'

print(person_info[0][0][0])  # 张

# 将 18 改为 19
person_info[0][1] = '19'
print(person_info)  # [['张三', '19', '功能测试'], ['李四', '20', '自动化测试']]

# 给 李四 所在的列表添加一个性别 信息
person_info[1].append('男')
print(person_info)  # [['张三', '19', '功能测试'], ['李四', '20', '自动化测试', '男']]

# 将张三的年龄信息删除
# person_info[0].pop(1)
person_info[0].remove('19')
print(person_info)  # [['张三', '功能测试'], ['李四', '20', '自动化测试', '男']]
```

元组

元组：tuple，元组的特点和列表非常相似

1. 元组中可以存放任意类型的数据
2. 元组中可以存放任意多个数据

区别：
1. 元组中的数据内容不能改变，列表中的可以改变的
2. 元组使用 ()，列表 使用 []

应用：在函数的传参或者返回值中使用，保证数据不会被修改

定义

1. 使用 类实例化的方式
2. 直接使用 () 方式

常用方法

由于元组中的数据不能修改,所以只有查看的方法

1. 在元组中也可以使用 下标和切片获取数据
2. 在元组中存在 index 方法, 查找下标, 如果不存在,会报错
3. 在元组中存在 count 方法, 统计数据出现的次数
4. 在元组中可以使用 in 操作, 判断数据是否存在
5. len() 统计个数

以上方法的使用 和列表中一样的

```
# 1, 类实例化的方式
# 1.1 定义空元祖(不会使用的)
my_tuple1 = tuple()
print(type(my_tuple1), my_tuple1)  # <class 'tuple'>
()
# 1.2 类型转换
# 可以将列表转换为元组, 只需要将 [], 变为 (), 同时 可以
将元组转换列表 , 将() 变为 []
my_tuple2 = tuple([1, 2, 3])
print(my_tuple2)  # (1, 2, 3)
# 转换字符串, 和列表中一样,只是将列表的[] 变为()
my_tuple3 = tuple('hello')
print(my_tuple3)  # ('h', 'e', 'l', 'l', 'o')

# 2. 直接使用 () 定义
my_tuple4 = (1, "小王", 3.14, False)
print(my_tuple4)
```

```
# 3. 特殊点，定义只有一个数据的元组时，数据后边必须有一个逗号
my_tuple5 = (1,)
print(my_tuple5)  # (1,)

print(my_tuple4[1])  # 小王
```

字典

1. 字典 dict，字典中的数据是由键(key)值(value)对组成的(键表示数据的名字，值就是具体的数据)
2. 在字典中一组键值对是一个数据，多个键值对之间使用 逗号隔开

变量 = {key: value, key:value, ...}

3. 一个字典中的键是唯一的,不能重复的，值可以是任意数据
4. 字典中的键 一般都是 字符串,可以是数字，不能是列表

定义

```
# 1. 使用 类实例化的方法
my_dict = dict()
print(type(my_dict), my_dict)  # <class 'dict'> {}
```

```
# dict() 不能转列表和元组,字符串

# 2. 直接使用{} 定义
# 2.1 空字典
my_dict1 = {}
print(type(my_dict1), my_dict1)  # <class 'dict'> {}

# 2.2 非空字典, 小明('name') 18('age') 1.71('height')
True(is_men)  抽烟 喝酒 烫头('like')
my_dict2 = {"name": "小明", "age": 18, "height":
1.71, "is_men": True, "like": ["抽烟", "喝酒", "烫
头"]}

print(my_dict2)
print(len(my_dict2))  # 5
```

增加和修改操作

```
语法:
字典[键] = 数据值
1. 如果键已经存在,就是修改数据值
2. 如果键不存在,就是添加数据(即添加键值对)
```

```
# 定义字典  小明 18 爱好
```

```python
my_dict = {"name": "小明", "age": 18, "like": ['抽烟', '喝酒', '烫头']}

print(my_dict)  # {'name': '小明', 'age': 18, 'like': ['抽烟', '喝酒', '烫头']}
# 1. 添加性别信息 sex
my_dict['sex'] = '男'
print(my_dict)  # {'name': '小明', 'age': 18, 'like': ['抽烟', '喝酒', '烫头'], 'sex': '男'}

# 2. 修改年龄为 19
my_dict['age'] = 19
print(my_dict)  # {'name': '小明', 'age': 19, 'like': ['抽烟', '喝酒', '烫头'], 'sex': '男'}

# 3. 添加一个爱好，学习--> 本质是向列表中添加一个数据
my_dict['like'].append('学习')
print(my_dict)  # {'name': '小明', 'age': 19, 'like': ['抽烟', '喝酒', '烫头', '学习'], 'sex': '男'}
```

删除

- 删除指定键值对

```
del 字典[键]
or
字典.pop(键)  # 键必须书写
```

- 清空

```
字典.clear()
```

```
my_dict = {'name': '小明', 'age': 19, 'like': ['抽烟', '喝酒', '烫头', '学习'], 'sex': '男'}

# 删除  sex 键值对
del my_dict['sex']
print(my_dict)  # {'name': '小明', 'age': 19, 'like': ['抽烟', '喝酒', '烫头', '学习']}

# 字典.pop('键')
my_dict.pop('age')
print(my_dict)  # {'name': '小明', 'like': ['抽烟', '喝酒', '烫头', '学习']}

# 删除抽烟的爱好 ---> 本质操作是在列表中删除 数据值
# my_dict['like'].pop(0)
```

```
my_dict['like'].remove('抽烟')
print(my_dict)  # {'name': '小明', 'like': ['喝酒', '烫头', '学习']}


# 清空键值对
my_dict.clear()
print(my_dict)  # {}
```

查询- 根据键获取对应的值

字典中没有下标的概念,想要获取数据值,要使用 key(键)来获取

- 使用 `字典[键]`

  字典[键]
  1. 如果键存在 返回键对应的数据值,
  2, 如果键不存在, 会报错

- 使用 `字典.get(键)`

字典.get(键, 数据值)

1. 数据值一般不写，默认是 None

返回:

1. 如果键存在 返回键对应的数据值

2, 如果键不存在，返回的是 括号中书写的数据值(None)

一般建议使用 get 方法

```
my_dict = {'name': '小明', 'age': 19, 'like': ['抽烟', '喝酒', '烫头', '学习']}

# 1. 获取 名字
print(my_dict['name'])  # 小明
print(my_dict.get('name'))  # 小明
print(my_dict.get('name', 'zzz'))  # 小明

# 2. 获取 sex 性别
# print(my_dict['sex'])   # 代码会报错，原因 key 不存在

print(my_dict.get('sex'))  # None
print(my_dict.get('sex', '保密'))  # 保密

# 3. 获取 第二个爱好
print(my_dict['like'][1])  # 喝酒
```

```
print(my_dict.get('like')[1])  # 喝酒
```

字典的遍历

对 字典的键 进行遍历

```
for 变量 in 字典:
    print(变量)  # 变量就是字典的 key, 键


for 变量 in 字典.keys():  # 字典.keys() 可以获取字典中所有的键
    print(变量)
```

对 字典的值 进行遍历

```
for 变量 in 字典.values():   # 字典.values() 可以获取字典中所有的值
    print(变量)
```

对 字典的键值对 进行遍历

```
# 变量1 就是 键，变量2 就是键对应的值
for 变量1, 变量2 in 字典.items():  # 字典.items() 获取键值对
    print(变量1, 变量2)
```

```
# 定义字典
my_dict = {'name': '小明', 'age': 18, 'sex': '男'}

# 1. 遍历字典的键
for k in my_dict:
    print(k)

for k in my_dict.keys():
    print(k)

print('-' * 30)

# 2. 遍历字典的值
for v in my_dict.values():
    print(v)

print('*' * 30)

# 3. 遍历键值
for k, v in my_dict.items():
```

```
    print(k, v)
```

容器部分 总结

```
# 1. 字符串, 列表, 元组 支持加法运算
str1 = 'hello' + ' world'  # 'hello world'
list1 = [1, 2] + [3, 4]  # [1, 2, 3, 4]
tuple1 = (1, 2) + (3, 4)  # (1, 2, 3, 4)

# 2. 字符串 列表 元组 支持 乘一个数字
'hello ' * 3 # ===> 'hello hello hello '
[1, 2] * 3  # ===> [1, 2, 1, 2, 1, 2]
(1, 2) * 3  # ===> (1, 2, 1, 2, 1, 2)

# 3. len() 在 容器中都可以使用

# 4. in 关键字在容器中都可以使用, 注意, 在字典中判断的是字典的键是否存在
```

函数

```
print()
input() ---> str
type()
```

概念

函数，就是把 具有独立功能的代码块 组织为一个小模块，在需要的时候 调用

函数,通俗理解， 将多行代码写在一块,起个名字， 在需要这多行代码的时候,可以直接使用这个名字来代替

函数好处：减少代码的冗余(重复的代码不用多写)，提高程序的编写效率

函数定义

1. 将多行代码放在一块,起名字的过程， 称为函数定义
2. 函数必须先定义后调用

- 语法

```
def 函数名():
    函数中的代码
    函数中的代码

# 1. def 是关键字，用来定义函数的  define 的缩写
# 2. 函数名需要遵守标识符的规则
# 3. 处于 def 缩进中的代码,称为函数体
# 4. 函数定义的时候，函数体中的代码不会执行，在调用的时候才会执行
```

- 函数定义小技巧

```
在前期,书写不熟练的时候,
1. 可以先不管函数,先把功能写出来,
2. 给多行代码起名字
3. 选中多行代码，使用 tab 键进行缩进
```

函数的调用

```
1. 使用多行代码的时候，称为函数调用
```

- 语法

```
函数名()

# 1. 函数调用的时候会执行函数体中代码
# 2. 函数调用的代码,要写在 函数体外边
```

```
def say_hello():
    print('hello 1')
    print('hello 2')
    print('hello 3')


# 调用
say_hello()

say_hello()
```

文档注释[了解]

文档注释的本质,还是注释，只不过书写的位置和作用比较特殊.

1. 书写位置，在函数名的下方使用 三对双引号进行的注释
2. 作用：告诉别人这个函数如何使用的，干什么的
3. 查看，在调用的时候，将光标放到函数名上,使用快捷键

Ctrl q(Windows)

Mac(ctrl j)

ctrl(cmd) B 转到函数声明中查看(按住Ctrl(cmd) 鼠标左键点击)

```
1   def say_hello():
2       """打印三个 hello"""
3       print('hello 1')
4       print('hello 2')
5       print('hello 3')
6
7
8   # 调用
9   say_hello()
0
1   say_hello()
2
```

hm_14_函数的使用
def say_hello() -> None 返回 None
打印三个 hello

函数的嵌套调用

在一个函数定义中调用另一个函数

1. 函数定义不会执行函数体中的代码
2. 函数调用会执行函数体中的代码
3. 函数体中代码执行结束会回到函数被调用的地方继续向下执行

```
def func1():
    print(1)
    print('func1')
    print(2)


def func2():
    print(3)
    func1()
    print(4)


print(5)
func2()
print(6)

# 5 3 1 2 4 6
```