day03 课堂笔记

课程之前

复习和反馈

作业

今日内容

```
数据序列 , 容器 字符串 str, 列表 list,元组 tuple, 字典 dict
了解 集合 set
```

字符串

定义

```
使用引号(单引号, 双引号, 三引号)引起来的内容,就是字符串
```

```
# 1. 使用单引号
str1 = 'hello'
# 2. 使用双引号定义
str2 = "hello"
# 3. 使用 三引号 定义
str3 = """hello"""
str4 = '''hello'''

print(type(str1), type(str2), type(str3), type(str4))

# 4. 定义字符串  I'm 小明, 字符串本身包含引号
# 4.1 如果字符串本身包含单引号,定义的时候不能使用 单引号,
# 4.2 如果字符串本身包含双引号,定义的时候不能使用 双引号,

str5 = "I'm 小明"
print(str5)  # I'm 小明

# 5. 转义字符 \n \t \' \"
str6 = 'I\'m 小明'
print(str6)  # I'm 小明

# 6. I\'m 小明  \\ --> \
str7 = 'I\\\'m 小明'
print(str7)  # I\'m 小明


# 7. 原生字符串 在字符串的前边 加上 r"", 字符串中的 \ 就不会进行转义
```

```
str8 = r'I\'m 小明'
print(str8)  # I\'m 小明

str7 = r'I\\\'m 小明'
print(str7)  # I\\\'m 小明
```

下标(索引)

```
1, 下标(索引), 是数据在容器(字符串, 列表, 元组)中的位置, 编号
2, 一般来说,使用的是正数下标, 从 0 开始
3, 作用: 可以通过下标来获取具体位置的数据. 使用的语法为
    容器[下标]
4, Python 中是支持负数下标, -1 表示最后一个位置的数据
```

0	1	2	3	4	5	6	正数下标(索引)
a	b	c	d	e	f	g	字符串
-7	-6	-5	-4	-3	-2	-1	负数下标

```
str1 = 'abcdefg'

# 需求: 打印输出字符串中的 a 字符
print(str1[0])
print(str1[-7])

# 需求: 打印输出字符串中 最后一个字符(-1)
print(str1[-1])

# 需求: 打印输出 下标为 3 的位置的字符
print(str1[3])
```

切片

```
1, 使用切片操作, 可以一次性获取容器中的多个数据(多个数据之间存在一定的规律,数据的下标是 等差数列(相邻的两个数字之间的差值是一样的))
2, 语法  容器[start:end:step]
2.1 start   表示开始位置的下标
2.2 end     表示结束位置的下标,但是 end 所对应的下标位置的数据是不能取到的
2.3 step    步长,表示的意思就是相邻两个坐标的差值
start, start+step, start+step*2, ...,end(取不到)
```

```
my_str = 'abcdefg'

# 需求1 : 打印字符串中 abc 字符 start 0, end 3, step 1
```

```
print(my_str[0:3:1])  # abc

# 1.1 如果步长是 1，可以省略不写
print(my_str[0:3])  # abc

# 1.2 如果 start 开始位置的下标为 0，可以不写,但是冒号不能少
print(my_str[:3])  # abc

# 需求 2: 打印字符串中的 efg ，start 4，end 7，step 1
print(my_str[4: 7])  # efg
# 2.1 如果取到最后一个字符，end  可以不写,但是冒号不能少
print(my_str[4:])  # efg

# 需求 3: 打印字符串中的 aceg ，start 0，end 7(最后)，步长 2
print(my_str[::2])  # aceg

# 练习: cf
print(my_str[2:6:3])

# 特殊情况，步长为 -1，反转(逆序) 字符串
print(my_str[::-1])   # gfedcba
```

字符串查找方法 find()

```
字符串.find(sub_str)   # 在字符串中 查找是否存在 sub_str 这样的字符串
返回值(这行代码执行的结果):
1，如果存在sub_str，返回 第一次出现 sub_str 位置的下标
2，如果不存在sub_str，返回 -1
```

```
# 1. 现有字符串数据: '黑马程序员'
# 2. 请设计程序，实现判断"黑马"和"白马"是否存在于数据中
# 3. 要求如果数据存在，则输出数据所在位置

my_str = '黑马程序员'
# sub_str = '黑马'
sub_str = '白马'
result = my_str.find(sub_str)
if result == -1:
    print(f'{sub_str}不存在')
else:
    print(f'{sub_str}存在，下标位置为:', result)
```

字符串的替换 replace()

字符串.replace(old, new, count) # 将字符串中的 old 字符串 替换为 new 字符串
- old 原字符串,被替换的字符串
- new 新字符串,要替换为的字符串
- count 一般不写,表示全部替换，可以指定替换的次数
- 返回：会返回一个替换后的完整的字符串
- 注意：原字符串不会改变的

```
my_str = 'good good study'

# 需求，将 good 变为 GOOD
my_str1 = my_str.replace('good', 'GOOD')
print('my_str :', my_str)
print('my_str1:', my_str1)

# 将第一个 good 替换为 Good
my_str2 = my_str.replace('good', 'Good', 1)
print('my_str2:', my_str2)

# 将第二个 good  替换为 Good
# 先整体替换为 Good，再将替换后的 第一个Good 替换为 good
my_str3 = my_str.replace('good', 'Good').replace('Good', 'good', 1)
print('my_str3:', my_str3)
```

字符串拆分 split()

字符串.split(sep) # 将字符串按照指定的字符串 sep 进行分隔
- sep ，按照 sep 分隔，可以不写，默认按照空白字符(空格 \t \n)分隔

返回：列表,列表中的每个数据就是分隔后的字符串

```
str1 = 'hello Python\tand itcast and\nitheima'

# 1. 默认 按照空白字符分隔
list1 = str1.split()
print(list1)  # ['hello', 'Python', 'and', 'itcast', 'and', 'itheima']

# 2. 按照 空格分隔
list2 = str1.split(' ')
print(list2)  # ['hello', 'Python\tand', 'itcast', 'and\nitheima']

# 3. 按照 and  分隔
list3 = str1.split('and')
print(list3)  # ['hello Python\t', ' itcast ', '\nitheima']
```

字符串的连接 join

字符串.join(容器) # 容器一般是列表 ，将字符串插入到列表相邻的两个数据之间,组成新的字符串
注意点：列表中的数据 必须都是字符串才可以

```
list1 = ['hello', 'Python', 'and', 'itcast', 'and', 'itheima']

# 将 列表中数据使用 空格 组成新的字符串
str1 = ' '.join(list1)
print(str1)  # hello Python and itcast and itheima

# 使用 逗号 连接
str2 = ','.join(list1)
print(str2)  # hello,Python,and,itcast,and,itheima

# 使用 _*_ 连接
str3 = '_*_'.join(list1)
print(str3)  # hello_*_Python_*_and_*_itcast_*_and_*_itheima
```

列表 list

定义

1，列表,list，使用 []
2，列表可以存放任意多个数据
3，列表中可以存放任意类型的数据
4，列表中数据之间 使用 逗号隔开

```
# 方式1，使用类实例化的方式
# 1.1 定义空列表  变量 = list()
list1 = list()
print(type(list1), list1)  # <class 'list'> []

# 1.2 定义非空列表 ，也称为 类型转换  list(可迭代类型)  可迭代类型,能够使用 for 循环 就是 可迭代类型(比如 容器)
# 将容器中的 每个数据 都作为列表中一个数据进行保存
list2 = list('abcd')
print(list2)  # ['a', 'b', 'c', 'd']


# 方式2，直接使用 [] 进行定义(使用较多)
# 2.1 定义空列表
list3 = []
print(list3)
# 2.2 定义非空列表
list4 = [1, 3.14, 'hello', False]
print(list4)
```

列表 支持下标 和 切片

列表的切片 得到是 新的列表.
字符串的切片 得到是 新的字符串

如果下标 不存在.会报错.

```
list4 = [1, 3.14, 'hello', False]
print(list4)

# 获取 列表中 第一个数据
print(list4[0])  # 1
# 获取列表中最后一个数据
print(list4[-1])  # False
# 获取中间两个数据即 3.14 和 'hello' (1 和 2)
print(list4[1: 3])  # [3.14, 'hello']
```

列表查询

index() 方法

index() 这个方法的作用和 字符串中的 find() 的作用是一样
列表中是没有 find() 方法的, 只有 index() 方法
字符串中 同时存在 find() 和 index() 方法

index()
1, 找到 返回下标
2, 没有找到, 直接报错

count() 方法

列表.count(数据) # 统计 指定数据在列表中出现的次数

```python
list1 = ['hello', 2, 3, 2, 3, 4]

# 查找 2 出现的下标
num = list1.index(2)
print(num)

# 统计数据 2 出现的次数
num1 = list1.count(2)
print(num1)

# 统计数据 20 出现的次数
num2 = list1.count(20)
print(num2)  # 0
```

添加数据 append() [重点]

```
列表.append(数据)  # 想列表的尾部添加数据
# 返回: None，所以不用使用 变量 = 列表.append()
直接在原列表中添加数据，不会生成新的列表,如果想要查看添加后的数据，直接 print() 打印原列表
```

删除数据 pop()

```
列表.pop(index)   # 根据下标删除列表中的数据
- index 下标可以不写，默认删除在最后一个
- 返回，删除的数据
```

```python
# 定义空列表
list1 = []

print(list1)
# 添加数据 张三
list1.append('张三')
print(list1)

# 添加李四
list1.append('李四')
print(list1)
list1.append('王五')
list1.append('赵六')
print(list1)

# 删除最后一个数据
list1.pop()
print(list1)
# 删除第二个数据
name = list1.pop(1)
print('删除的对象为:', name)
print(list1)
```

修改数据

想要修改列表中的数据，直接是所有下标即可

列表[下标] = 新数据

```
my_list = [1, 2]

my_list[0] = 10
print(my_list)

my_list[-1] = 200
print(my_list)
```

列表的反转 reverse()

字符串 反转 字符串[::-1]

列表 反转
1. 列表[::-1] 得到一个新的列表，原列表不会改动
2. 列表.reverse() 直接修改原列表的数据

```
my_list = ['a', 'b', 'c', 'd', 'e']

# 1. 切片
my_list1 = my_list[::-1]
print('my_list :', my_list)
print('my_list1:', my_list1)


# 2. reverse
my_list.reverse()
print('my_list :', my_list)
```

列表的排序

```
# 前提：列表中的数据要一样
列表.sort()  # 升序，从小到大，直接在原列表中进行排序
列表.sort(reverse=True) # 降序，从大到下，直接在原列表中进行排序
```

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

# 排序 升序
my_list.sort()
print(my_list)

# 降序
my_list.sort(reverse=True)
print(my_list)
```

列表的嵌套

列表的嵌套 就是指 列表中数据都是列表.

```python
student_list = [["张三", "18", "功能测试"], ["李四", "20", "自动化测试"], ["王五", "21", "自动化测试"]]

# 张三
print(student_list[0][0])
# 李四
print(student_list[1][0])

# 张三 的信息添加一个 性别 男 ---> 向张三所在的列表 添加数据
student_list[0].append('男')
print(student_list)
# 删除 性别
student_list[0].pop()
print(student_list)

# 打印 所有人员的年龄
for info in student_list:  # info 是 列表
    print(info[1])
```

元组 tuple

1, 元组 tuple, 使用的 ()
2, 元组和列表非常相似, 都可以存储多个数据, 都可以存储任意类型的数据
3, 区别就是 元组中的数据不能修改,列表中可以修改
4, 因为元组中的数据不能修改,所以只能 查询方法, 如 index, count ,支持下标和切片
5, 元组, 主要用于传参和返回值

```python
# 1. 类实例化方式

# 1.1 定义空元组(不用)
```

```
tuple1 = tuple()
print(type(tuple1), tuple1)  # <class 'tuple'> ()
# 1.2 类型转换 , 将列表(其他可迭代类型)转换为元组
tuple2 = tuple([1, 2, 3])
print(tuple2)

# 2. 直接使用 () 定义
# 2.1 定义空元组
tuple3 = ()
# 2.2 非空元组
tuple4 = (1, 2, 'hello', 3.14, True)
print(tuple4)

print(tuple4[2])  # hello

# 2.3 定义只有一个数据的元组， 数据后必须有一个逗号
tuple5 = (10,)
print(tuple5)
```

应用- 交换两个变量的值

```
1, 在定义元组的时候，小括号可以省略不写
2, 组包(pack), 将多个数据值组成元组的过程 a = 1, 2  # a = (1, 2)
3, 拆包(解包 unpack), 将容器中多个数据 分别给到多个变量，需要保证容器中元素的个数和变量 的个数保持一致
```

```
a = 10
b = 20
# c = b, a  # 组包
# print(c)  # (20, 10)
# a, b = c  # 拆包 a(20)  b(10)
# print(a, b)

a, b = b, a
print(a, b)

x, y, z = 'abc'
print(y)  # b
```

字典 dict

定义

```
1，字典 dict，使用 {}  表示
2，字典是由键(key)值(value)对组成的，  key: value
3，一个键值对是一组数据，多个键值对之间使用 逗号隔开
4，在一个字典中，字典的键 是不能重复的
5，字典中的键 主要使用 字符串类型，可以是数字
6，字典中没有下标
```

```python
# 1，类实例化的方式
my_dict1 = dict()
print(type(my_dict1), my_dict1)  # <class 'dict'> {}

# 2，直接使用 {}  定义
# 2.1 定义空字典
my_dict2 = {}
print(my_dict2)

# 2.2 定义非空字典，姓名，年龄，身高，性别
my_dict = {"name": "小明", "age": 18, "height": 1.78, "isMen": True}
print(my_dict)
```

增加和修改

```
字典['键'] = 值
# 1，键 存在，修改
# 2，键 不存在，添加
```

```python
定义非空字典，姓名，年龄，身高，性别
my_dict = {"name": "小明", "age": 18, "height": 1.78, "isMen": True}
print(my_dict)

# 将年龄改为 20
my_dict['age'] = 20
print(my_dict)

# 添加 体重 weight
my_dict['weight'] = 65
print(my_dict)
```

删除

```
字典的删除是根据字典的键 删除键值对
字典.pop('键')
```

```
my_dict.pop('weight')
print(my_dict)
my_dict.pop('height')
print(my_dict)
```

查询

```
根据字典的 键, 获取对应的 值.

方法一:
字典['键']   # 键 不存在,会报错

# 方法 二
字典.get(键)   # 键不存在,返回 None
```

```
my_dict = {'name': '小明', 'age': 20}

# 获取 name 值
print(my_dict['name'])
print(my_dict.get('name'))

# 获取 性别 sex
# print(my_dict['sex'])   # 会报错, 因为 键不存在
print(my_dict.get('sex'))
```

遍历

```
字典存在 键(key), 值(value) , 遍历分为三种情况
```

遍历字典的键

```
# 方式一
for 变量 in 字典:
    print(变量)

# 方式二
for 变量 in 字典.keys():  # 字典.keys() 可以获取字典所有的键
    print(变量)
```

遍历字典的值[使用较多]

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

遍历字典的键和值

```
# 变量1 就是 键， 变量2 就是值
for 变量1, 变量2 in 字典.items():  #  字典.items() 获取的是字典的键值对
    print(变量1, 变量2)
```

```
my_dict = {'name': '小明', 'age': 18, 'sex': '男'}

for k in my_dict:
    print(k)

print('*' * 30)
for k in my_dict.keys():
    print(k)

print('-' * 30)
for v in my_dict.values():
    print(v)

print('_*_' * 30)
for k, v in my_dict.items():
    print(k, v)
```