第八次课

一 字符串定义

学习字符串之前必须记住一句话：字符串一旦创建，永远不能被修改

字符串 就是 **一串字符**，是编程语言中表示**文本**的数据类型

在 Python 中可以使用 **一对双引号** " 或者 **一对单引号** ' 定义一个**字符串**

虽然可以使用 \" 或者 \' 做字符串的转义，但是在实际开发中：

如果字符串内部需要使用 "，可以使用 ' 定义字符串

如果字符串内部需要使用 '，可以使用 " 定义字符串

可以使用 **索引** 获取一个字符串中 **指定位置**的字符，索引计数**从 0** 开始

也可以使用 **for 循环遍历** 字符串中每一个字符

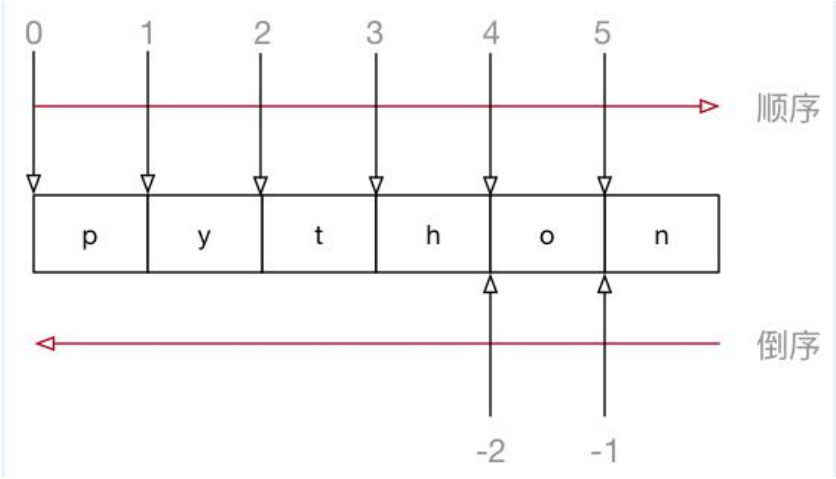
大多数编程语言都是用 " 来定义字符串

```
string = "Hello Python"

for c in string:

    print(c)
```

字符串的索引示意图如下所示：



二 字符串的常用操作

在 python3 中定义一个 **字符串**，例如：hello_str = ""

字符串 能够使用的 方法 如下：

```
In [1]: hello_str.

hello_str.capitalize    hello_str.isidentifier  hello_str.rindex

hello_str.casefold      hello_str.islower       hello_str.rjust

hello_str.center        hello_str.isnumeric     hello_str.rpartition

hello_str.count         hello_str.isprintable   hello_str.rsplit

hello_str.encode        hello_str.isspace       hello_str.rstrip

hello_str.endswith      hello_str.istitle       hello_str.split

hello_str.expandtabs    hello_str.isupper       hello_str.splitlines

hello_str.find          hello_str.join          hello_str.startswith

hello_str.format        hello_str.ljust         hello_str.strip

hello_str.format_map    hello_str.lower         hello_str.swapcase

hello_str.index         hello_str.lstrip        hello_str.title

hello_str.isalnum       hello_str.maketrans     hello_str.translate

hello_str.isalpha       hello_str.partition     hello_str.upper

hello_str.isdecimal     hello_str.replace       hello_str.zfill

hello_str.isdigit       hello_str.rfind
```

提示：

正是因为 python **内置提供**的方法**足够多**，才使得在开发时，能够针对字符串进行更加灵活的操作！应对更多的开发需求！

2.1 判断类型 - 9

方法	说明
string.isspace()	如果 string 中只包含空格，则返回 True
string.isalnum()	如果 string 至少有一个字符并且所有字符都是字母或数字则返回 True
string.isalpha()	如果 string 至少有一个字符并且所有字符都是字母则返回 True
string.isdecimal()	如果 string 只包含数字则返回 True，全角数字

方法	说明
string.isdigit()	如果 string 只包含数字则返回 True，全角数字、⑴、\u00b2
string.isnumeric()	如果 string 只包含数字则返回 True，全角数字，汉字数字
string.istitle()	如果 string 是标题化的(每个单词的首字母大写)则返回 True
string.islower()	如果 string 中包含至少一个区分大小写的字符，并且所有这些(区分大小写的)字符都是小写，则返回 True
string.isupper()	如果 string 中包含至少一个区分大小写的字符，并且所有这些(区分大小写的)字符都是大写，则返回 True

注意：

string.isdecimal()、string.isdigit() 和 string.isnumeric()

咋一看，这三个方法都是判断字符串是否只包含数字，但是其实它们还是有很大的区别的：

1、都能判断 阿拉伯数字：

判断字符串是否只包含数字

```
# 判断字符串是否只包含数字

num_str = "1"

print(num_str)

print(num_str.isdecimal())

print(num_str.isdigit())

print(num_str.isnumeric())
```

结果：

```
1

True

True

True
```

2、都不能判断小数：

```
# 判断字符串是否只包含数字

num_str = "1.1"

print(num_str)

print(num_str.isdecimal())

print(num_str.isdigit())
```

```
print(num_str.isnumeric())
```

结果：

```
1.1
False
False
False
```

3、与 **isdecimal()** 相比，**isdigit() 和 isnumeric()** 都能判断 **unicode** 编码的数字字符串：如⑴、①和 \u00b2

```
# 判断字符串是否只包含数字
num_str = "\u00b2"
print(num_str)
print(num_str.isdecimal())
print(num_str.isdigit())
print(num_str.isnumeric())

num_str_1 = "⑴"
print(num_str_1)
print(num_str_1.isdecimal())
print(num_str_1.isdigit())
print(num_str_1.isnumeric())

num_str_2 = "①"
print(num_str_2)
print(num_str_2.isdecimal())
print(num_str_2.isdigit())
print(num_str_2.isnumeric())
```

结果：

```
²
False
True
```

```
True
⑴
False
True
True
①
False
True
True
```

4、与 isdigit() 相比， isnumeric() 还能判断 中文数字：如 一千一百零一

```
# 判断字符串是否只包含数字
num_str = "一千一百零一"
print(num_str)
print(num_str.isdecimal())
print(num_str.isdigit())
print(num_str.isnumeric())
```

结果：

```
一千一百零一
False
False
True
```

```
# 判断字符串是否只包含数字
num_str = "壹贰叁肆伍陆柒捌玖拾佰仟萬亿"
print(num_str)
print(num_str.isdecimal())
print(num_str.isdigit())
print(num_str.isnumeric())
```

结果：壹贰叁肆伍陆柒捌玖拾佰仟萬亿

```
False
False
```

```
True
```

结论：

简单一句话来说，对于数字的定义来说，**广义度：`isnumeric()` > `isdigit()` > `isdecimal()`**。

在 开发中**最常用**的还是仅**判断阿拉伯数字**的 **`isdecimal()`** 方法。

2.2 查找和替换 - 7

方法	说明
string.startswith(str)	检查字符串是否是以 str 开头，是则返回 True
string.endswith(str)	检查字符串是否是以 str 结束，是则返回 True
string.find(str, start=0, end=len(string))	检测 str 是否包含在 string 中，如果 start 和 end 指定范围，则检查是否包含在指定范围内，如果是返回开始的索引值，否则返回 -1
string.rfind(str, start=0, end=len(string))	类似于 find()，不过是从右边开始查找
string.index(str, start=0, end=len(string))	跟 find() 方法类似，不过如果 str 不在 string 会报错
string.rindex(str, start=0, end=len(string))	类似于 index()，不过是从右边开始
string.replace(old_str, new_str, num=string.count(old))	把 string 中的 old_str 替换成 new_str，如果 num 指定，则替换不超过 num 次

2.3 大小写转换-5

方法	说明
string.capitalize()	把字符串的第一个字符大写
string.title()	把字符串的每个单词首字母大写
string.lower()	转换 string 中所有大写字符为小写
string.upper()	转换 string 中的小写字母为大写
string.swapcase()	翻转 string 中的大小写

2.4 文件对齐- 3

方法	说明
string.ljust(width)	返回一个原字符串左对齐，并使用空格填充至长度 width 的新字符串
string.rjust(width)	返回一个原字符串右对齐，并使用空格填充至长度 width 的新字符串
string.center(width)	返回一个原字符串居中，并使用空格填充至长度 width 的新字符串

2.5 去除空白字符 - 3

方法	说明
string.lstrip()	截掉 string 左边（开始）的空白字符
string.rstrip()	截掉 string 右边（末尾）的空白字符
string.strip()	截掉 string 左右两边的空白字符

2.6 拆分和连接 - 5

方法	说明
string.partition(str)	把字符串 string 分成一个 3 元素的元组 (str 前面, str, str 后面)
string.rpartition(str)	类似于 partition() 方法，不过是从右边开始查找
string.split(str="", num)	以 str 为分隔符拆分 string，如果 num 有指定值，则仅分隔 num + 1 个子字符串，str 默认包含 '\r', '\t', '\n' 和空格
string.splitlines()	按照行('\r', '\n', '\r\n')分隔，返回一个包含各行作为元素的列表
string.join(seq)	以 string 作为分隔符，将 seq 中所有的元素（的字符串表示）合并为一个新的字符串

三 字符串的切片

切片 方法适用于 **字符串、列表、元组**

切片 使用 **索引值** 来限定范围，从一个大的 **字符串** 中 **切出** 小的 字符串

列表 和 **元组** 都是 **有序** 的集合，都能够 通过**索引值** 获取到对应的数据

字典 是一个 **无序** 的集合，是使用 **键值对** 保存数据

```
字符串[开始索引:结束索引:步长]
```

注意：

1. 指定的区间属于 **左闭右开** 型 [开始索引, 结束索引) => 开始索引 >= 范围 < 结束索引

   从 **起始** 位开始，到 **结束位的前一位** 结束（不包含结束位本身)

2. 从头开始，**开始索引 数字可以省略，冒号不能省略**

3. **到末尾结束，结束索引** 数字可以省略，**冒号不能省略**

4. **步长默认为 1**，如果连续切片，**数字和冒号都可以省略**

索引的顺序和倒序

在 Python 中不仅支持 **顺序索引**，同时还支持 **倒序索引**

所谓倒序索引就是 **从右向左** 计算索引

最右边的索引值是 **-1**，依次递减

演练需求

① 截取从 2 ~ 5 位置 的字符串

② 截取从 2 ~ 末尾 的字符串

③ 截取从 开始 ~ 5 位置 的字符串

④ 截取完整的字符串

⑤ 从开始位置，每隔一个字符截取字符串

⑥ 从索引 1 开始，每隔一个取一个

⑦ 截取从 2 ~ 末尾 - 1 的字符串

⑧ 截取字符串末尾两个字符

⑨ 字符串的逆序（面试题）

答案：

```
num_str = "0123456789"

# 1. 截取从 2 ~ 5 位置 的字符串

print(num_str[2:6])

# 2. 截取从 2 ~ `末尾` 的字符串

print(num_str[2:])

# 3. 截取从 `开始` ~ 5 位置 的字符串
```

```
print(num_str[:6])

# 4. 截取完整的字符串

print(num_str[:])

# 5. 从开始位置，每隔一个字符截取字符串

print(num_str[::2])

# 6. 从索引 1 开始，每隔一个取一个

print(num_str[1::2])

# 倒序切片# -1 表示倒数第一个字符

print(num_str[-1])

# 7. 截取从 2 ~ `末尾 - 1` 的字符串

print(num_str[2:-1])

# 8. 截取字符串末尾两个字符

print(num_str[-2:])

# 9. 字符串的逆序（面试题）

print(num_str[::-1])
```

四 补充知识点

4.1 Python 的内置函数

Python 包含了以下内置函数：

函数	描述	备注
len(item)	计算容器中元素个数	
del(item)	删除变量	del 有两种方式
max(item)	返回容器中元素最大值	如果是字典，只针对 key 比较
min(item)	返回容器中元素最小值	如果是字典，只针对 key 比较
cmp(item1, item2)	比较两个值，-1 小于/0 相等/1 大于	Python 3.x 取消了 cmp 函数

注意

字符串 比较符合以下规则： "0" < "A" < "a"

表 1-1 ASCII 编码表（片段）

码值	字符	码值	字符	码值	字符	码值	字符	码值	字符
48	0	63	?	78	N	93	]	108	l
49	1	64	@	79	O	94	^	109	m
50	2	65	A	80	P	95	_	110	n
51	3	66	B	81	Q	96	`	111	o
52	4	67	C	82	R	97	a	112	p
53	5	68	D	83	S	98	b	113	q
54	6	69	E	84	T	99	c	114	r
55	7	70	F	85	U	100	d	115	s
56	8	71	G	86	V	101	e	116	t
57	9	72	H	87	W	102	f	117	u
58	:	73	I	88	X	103	g	118	v
59	;	74	J	89	Y	104	h	119	w
60	<	75	K	90	Z	105	i	120	x
61	=	76	L	91	[	106	j	121	y
62	>	77	M	92	\	107	k	122	z

4.2 补充运算符

运算符	Python 表达式	结果	描述	支持的数据类型
+	[1, 2] + [3, 4]	[1, 2, 3, 4]	合并	字符串、列表、元组
*	["Hi!"] * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	重复	字符串、列表、元组
in	3 in (1, 2, 3)	True	元素是否存在	字符串、列表、元组、字典
not in	4 not in (1, 2, 3)	True	元素是否不存在	字符串、列表、元组、字典
> >= == < <=	(1, 2, 3) < (2, 2, 3)	True	元素比较	字符串、列表、元组

注意

in 在对 **字典** 操作时，判断的是 **字典的键**

in 和 **not in** 被称为 **成员运算符**

成员运算符

成员运算符用于 **测试** 序列中是否包含指定的 **成员**

运算符	描述	实例
in	如果在指定的序列中找到值返回 True，否则返回 False	`3 in (1, 2, 3)` 返回 `True`
not in	如果在指定的序列中没有找到值返回 True，否则返回 False	`3 not in (1, 2, 3)` 返回 `False`

4.3 完整的 for 循环语法

在 Python 中完整的 **for 循环** 的语法如下

```
for 变量 in 集合:

    循环体代码

else:

    没有通过 break 退出循环，循环结束后，会执行的代码
```

应用场景

在 **迭代遍历 嵌套**的数据类型时，例如 **一个列表包含了多个字典**

需求：要判断 某一个字典中 **是否存在** 指定的 **值**

如果 存在，提示并且退出循环

如果 不存在，在 循环整体结束 后，希望 得到一个统一的提示

```
students = [
    {"name": "阿土",
     "age": 20,
     "gender": True,
     "height": 1.7,
     "weight": 75.0},
    {"name": "小美",
     "age": 19,
     "gender": False,
     "height": 1.6,
     "weight": 45.0},
]
find_name = "阿土"
for stu_dict in students:
    print(stu_dict)
    # 判断当前遍历的字典中姓名是否为find_name
    if stu_dict["name"] == find_name:
        print("找到了")
        # 如果已经找到，直接退出循环，就不需要再对后续的数据进行比较
        break
else:
    print("没有找到")
print("循环结束")
```