程序设计（Python）

字符串及第三方库

主讲：数据与目标工程学院 胡瑞娟 副教授

模拟神州二十一号飞船对接空间站。

思考重点：需要几个数据？用**什么变量名**来表示数据？**数据的类型**是？（提问）

数据的来源是（固定值、**从键盘输入**……..）？

- Python数据类型分为：
- **基本数据类型**：
  - 数字类型（整数（int）、浮点数（float））
  - 布尔类型（bool）
  - 字符串类型(str)
- **组合数据类型**：
  - 集合、元组、列表、字典

类型	示例
数字	1234, 3.14, 3+4j
字符串	'swfu', "I'm student", '"Python "'
布尔型	True, False
集合	set('abc'), {'a', 'b', 'c'}
元组	(2, -5, 6, '3')
列表	[1, 2, 3, '3']
字典	{1:'food' ,2:'taste', '3':'import'}
文件	f=open('data.dat', 'r')
空类型	None
编程单元类型	函数、模块、类

案例引入

【案例】在军事通信中，信息的格式是生命线。一个格式错误的坐标、一条含义模糊的指令，可能导致任务失败。今天，我们的任务就是扮演一名情报分析员，处理混乱的原始战场数据，并**将其格式化为清晰、标准的战术指令(情报摘要)**。具体来说：
你截获了多段来自不同单位的战场通信文本。这些文本格式混乱、包含冗余代码和干扰字符。如：**"ALPHA7|n39-52.783,e116-25.467|T0830|CONFIRMED"**
你的任务是：

- 清洗和标准化文本，去除干扰符号
- 提取关键信息（单位代号、坐标、时间、状态）
- 将坐标格式标准化
- 生成清晰的情报摘要

字符串

1.字符串概念 - 认识什么是字符串
2.字符串长度 - 了解信息规模
3.字符串分割 - 拆解复杂信息
4.字符串查找/切片 - 定位关键内容
5.字符串替换 - 清洗和标准化格式
6.字符串拼接/输出 - 生成最终报告

01

字符串

字符串是由字母、符号或者数字组成的字符序列。

- **单引号**字符串：　' 单引号表示，可以使用"双引号"作为字符串的一部分 '
- **双引号**字符串：　"双引号表示，可以使用 ' 单引号 ' 作为字符串的一部分"
- **三引号**字符串：　' ' ' 三引号表示，可以使用"双引号" ' 单引号 ' 也可以使用换行 ' ' '

- 打印字符串的Python运行结果如下，注意其中的引号部分：

```
>>> print('单引号表示可以使用"双引号"作为字符串的一部分')
单引号表示可以使用"双引号"作为字符串的一部分
>>> print("双引号表示可以使用'单引号'作为字符串的一部分")
双引号表示可以使用'单引号'作为字符串的一部分
>>> print('''三引号中可以使用"双引号"
'单引号'
也可以使用换行''')
三引号中可以使用"双引号"
'单引号'
也可以使用换行
```

1.1字符串概念

```
>>> print('let's learn python')
SyntaxError: invalid syntax
```

```
print('let\'s learn Python')
```

示例

```
>>> print('let\'s learn python')
let's learn python
```

Python使用反斜杠“\”**转义**。例如，在字符串中的引号前添加“\”，此时Python解释器会将“\”之后的引号视为解释为一个普通字符，而非特殊符号。

```
print('转义字符中:\n表示换行;\r表示回车;\b表示退格')
```

示例

```
>>> print('转义字符中:\n表示换行;\r表示回车;\b表示退格')
转义字符中:
表示换行;表示回车;□表示退格
```

```
print('转义字符中:\\n表示换行;\\r表示回车;\\b表示退格')
```

输出原始字符串：在字符串开始的引号之前添加r或R，使它成为原始字符串。

```
print(r'转义字符中:\n表示换行;\r表示回车;\b表示退格')
```

示例

```
>>> print(r'转义字符中:\n表示换行;\r表示回车;\b表示退格')
转义字符中:\n表示换行;\r表示回车;\b表示退格
```

【案例分析】

问题1：接收一条加密信息，打印输出。

"ALPHA7|n39-52.783,e116-25.467|T0830|CONFIRMED"

```
message = "ALPHA7|n39-52.783,e116-25.467|T0830|CONFIRMED"
print("原始通信：", message)
```

如果有小写字母，转换为大写。

```
message = message.upper()
print("转换为大写后：", message)
```

问题2：字符串长度 - 了解信息规模

```
message_length = len(message)
print("信息长度：", message_length, "个字符")
```

1.2 字符串函数

- Python解释器提供了一些内置函数，其中与字符串处理相关的函数如下表。

函数	描述
str.lower()	将字符串str转换为小写
str.upper()	将字符串str转换为大写
str.swapcase()	将小写字母转换为大写，将大写字母转换为小写
str.capitalize()	仅将字符串的首字母转换为大写，其他字母转换为小写
str.title()	将字符串中每个单词的首字母大写，其余字母转换为小写

```
原始信息：  ALPHA7|n39-52.783,e116-25.467|T0830|CONFIRMED
转换为大写后：  ALPHA7|N39-52.783,E116-25.467|T0830|CONFIRMED
```

1.2 字符串函数

- Python解释器提供了一些内置函数，其中与字符串处理相关的函数如下表。

函数	描述
len(x)	返回字符串x的长度
str(x)	返回任意类型x所对应的字符串形式
chr(x)	返回Unicode编码x对应的单字符
ord(x)	返回单字符表示的Unicode编码
hex(x)	返回整数x对应十六进制数的小写形式字符串
oct(x)	返回整数x对应八进制数的小写形式字符串

Unicode编码则是采用双字节16位来进行编号，可编65536字符，基本上包含了世界上所有的语言字符

1.2 字符串函数

- Python解释器提供了一些内置函数，其中与字符串处理相关的函数示例。

```
>>> len('abcdefg')
7
>>> str(123123)
'123123'
>>> int('123')
123
>>> float('123213.2')
123213.2
>>> int(1231.1)
1231
>>> float(123)
123.0
```

```
>>> ord('a')
97
>>> chr(97)
'a'
>>> ord('中')
20013
>>> chr(20022)
'丶'
>>> chr(20000)
'北'
>>> oct(11)
'0o13'
```

【案例分析】

问题3：字符串分割 - 拆解复杂信息。

"ALPHA7|N39-52.783,E116-25.467|T0830|CONFIRMED"

```
分割后的信息块： ['ALPHA7', 'N39-52.783,E116-25.467', 'T0830', 'CONFIRMED']
信息块数量： 4
```

str.split() - 按指定分隔符分割字符串

```
message_parts = message.split('|') #这条信息用竖线 | 分隔成多个信息块
print("分割后的信息块：", message_parts)
print("信息块数量：", len(message_parts)
```

【案例分析】

问题3：字符串分割 - 拆解复杂信息。

"ALPHA7|N39-52.783,E116-25.467|T0830|CONFIRMED"
"ALPHA7|N39-52.783,E116-25.467|T0830|CONFIRMED "

这里多了个空格，如何去除?

str.strip() - 去除开头和结尾的空字符串

```
message_parts = message.strip(' ').split('|') #先去除空格再分割块
print("分割后的信息块：", message_parts)
print("信息块数量：", len(message_parts)
```

【案例分析】

问题3：字符串分割 - 拆解复杂信息。进一步思考：

"ALPHA7|N39-52.783,E116-25.467|T0830|CONFIRMED"

"ALPHA7|N39-52.783,E116-25.467*T0830&CONFIRMED "

如果分隔符不仅有'|',还有'*'、'&',如何分割?

使用正则表达式模块 re
re.split(r'[|*&]+',str)

```
import re
message_parts = re.split(r'[|*&]+',message) #分割包含多种分隔符的字符串, 使用正则表达式模块 re 可以实现复杂的分隔符分割
print("分割后的信息块：", message_parts)
print("信息块数量：", len(message_parts)
```

【案例分析】

问题4：字符串查找 - 定位关键内容。

"ALPHA7|N39-52.783,E116-25.467|T0830|CONFIRMED"

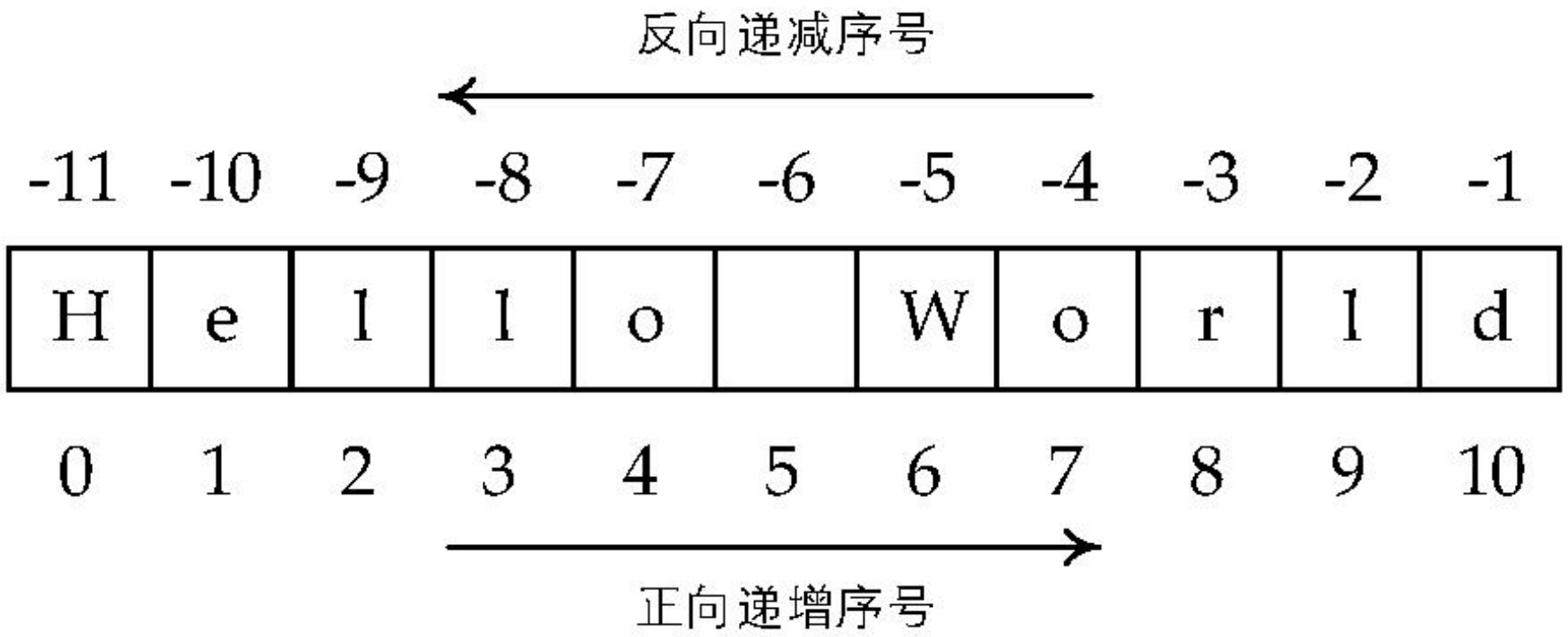
```
分割后的信息块： ['ALPHA7', 'N39-52.783,E116-25.467', 'T0830', 'CONFIRMED']
信息块数量： 4
```

```
# 提取各个信息块
unit_code = message_parts[0]      # 单位代号
coordinates = message_parts[1]    # 坐标信息
time_stamp = message_parts[2]     # 时间戳
status = message_parts[3]         # 状态
```

```
单位名称： ALPHA
单位编号： 7
纯时间： 0830
=== 坐标信息处理 ===
逗号位置： 10
短横线位置： 3
标准化坐标： N3952.783,E11625.467
逗号位置： 9
纬度部分： N3952.783
经度部分： E11625.467
纬度方向： N
纬度数值： 3952.783
```

字符串访问（查找/切片）

- 字符串包括两种序号体系：**正向递增**序号和**反向递减**序号。



- **区间访问（切片）**方式：采用**[N: M]**格式，表示字符串中从N到M（不包含M）的子字符串，其中，N和M为字符串的索引序号，可以混合使用正向递增序号和反向递减序号。如果表示中M或者N索引缺失，则表示字符串把开始或结束索引值设为默认值。

string[开始索引值：结束索引值：步长（默认1）]

整个序列：string 或者string[:]

1.3 字符串访问

```
>>> x='0123456789'
>>> x[1:7:2]          '135'
>>> x[::2]            '02468'
>>> x[7:1:-2]         '753'          #返回偏移量为7、5、3的字符
>>> x[::-1]           '9876543210'#将字符串反序返回
```

- 填空题

```
userString = 'aabbcc'

subStr1 = userString[2:]
subStr2 = userString[1:4]
subStr3 = userString[:-2]
idxInt = 0

print(subStr1) [填空1]    bbcc
print(subStr2) [填空2]    abb
print(subStr3) [填空3]    aabb
print(userString[idxInt+2]) [填空4]   b
```

【案例分析】

问题4：字符串查找/切片 - 定位关键内容。

"ALPHA7|N39-52.783,E116-25.467|T0830|CONFIRMED"

```
分割后的信息块： ['ALPHA7', 'N39-52.783,E116-25.467', 'T0830', 'CONFIRMED']
信息块数量： 4
```

```
# 提取各个信息块
unit_code = message_parts[0]      # 单位代号
coordinates = message_parts[1]    # 坐标信息
time_stamp = message_parts[2]     # 时间戳
status = message_parts[3]         # 状态
```

```
unit_name = unit_code[0:5]  # 从索引0开始，到索引5之前（不包含5）
unit_number = unit_code[5]  # 只取索引5的字符
print("单位名称：", unit_name)
print("单位编号：", unit_number)
```

```
单位名称： ALPHA
单位编号： 7
纯时间： 0830
=== 坐标信息处理 ===
逗号位置： 10
短横线位置： 3
标准化坐标： N3952.783,E11625.467
逗号位置： 9
纬度部分： N3952.783
经度部分： E11625.467
纬度方向： N
纬度数值： 3952.783
```

【案例分析】

问题4：字符串查找/切片 - 定位关键内容。

"ALPHA7|N39-52.783,E116-25.467|T0830|CONFIRMED"

```
分割后的信息块： ['ALPHA7', 'N39-52.783,E116-25.467', 'T0830', 'CONFIRMED']
信息块数量： 4
```

```
# 提取各个信息块
unit_code = message_parts[0]      # 单位代号
coordinates = message_parts[1]    # 坐标信息
time_stamp = message_parts[2]     # 时间戳
status = message_parts[3]         # 状态
```

```
pure_time = time_stamp[1:]  # 从索引1开始
print("纯时间： ", pure_time)
```

```
单位名称： ALPHA
单位编号： 7
纯时间： 0830
=== 坐标信息处理 ===
逗号位置： 10
短横线位置： 3
标准化坐标： N3952.783,E11625.467
逗号位置： 9
纬度部分： N3952.783
经度部分： E11625.467
纬度方向： N
纬度数值： 3952.783
```

【案例分析】

问题4：字符串查找/切片 - 定位关键内容。

"ALPHA7|N39-52.783,E116-25.467|T0830|CONFIRMED"

```
分割后的信息块： ['ALPHA7', 'N39-52.783,E116-25.467', 'T0830', 'CONFIRMED']
信息块数量： 4
```

```
# 提取各个信息块
unit_code = message_parts[0]      # 单位代号
coordinates = message_parts[1]    # 坐标信息
time_stamp = message_parts[2]     # 时间戳
status = message_parts[3]         # 状态
```

```
print("=== 坐标信息处理 ===")
comma_position = coordinates.find(',')# find()- 查找字符位置
dash_position = coordinates.find('-')
print("逗号位置：", comma_position)
print("短横线位置：", dash_position)
```

```
单位名称： ALPHA
单位编号： 7
纯时间： 0830
=== 坐标信息处理 ===
逗号位置： 10
短横线位置： 3
标准化坐标： N3952.783,E11625.467
逗号位置： 9
纬度部分： N3952.783
经度部分： E11625.467
纬度方向： N
纬度数值： 3952.783
```

【案例分析】

问题5：字符串替换 - 清洗和标准化格式。

"ALPHA7|N39-52.783,E116-25.467|T0830|CONFIRMED"

```
分割后的信息块： ['ALPHA7', 'N39-52.783,E116-25.467', 'T0830', 'CONFIRMED']
信息块数量： 4
```

```
# 提取各个信息块
unit_code = message_parts[0]      # 单位代号
coordinates = message_parts[1]    # 坐标信息
time_stamp = message_parts[2]     # 时间戳
status = message_parts[3]         # 状态
```

```
单位名称： ALPHA
单位编号： 7
纯时间： 0830
=== 坐标信息处理 ===
逗号位置： 10
短横线位置： 3
标准化坐标： N3952.783,E11625.467
逗号位置： 9
纬度部分： N3952.783
经度部分： E11625.467
纬度方向： N
纬度数值： 3952.783
```

```
standard_coords = coordinates.replace('-', '')
#replace() - 替换字符
print("标准化坐标： ", standard_coords)
comma_position = standard_coords.find(',')
print("逗号位置： ", comma_position)
```

【案例分析】

问题4：字符串查找 /切片- 定位关键内容。

"ALPHA7|N39-52.783,E116-25.467|T0830|CONFIRMED"

```
分割后的信息块： ['ALPHA7', 'N39-52.783,E116-25.467', 'T0830', 'CONFIRMED']
信息块数量： 4
```

```
# 提取各个信息块
unit_code = message_parts[0]      # 单位代号
coordinates = message_parts[1]    # 坐标信息
time_stamp = message_parts[2]     # 时间戳
status = message_parts[3]         # 状态
```

```
单位名称： ALPHA
单位编号： 7
纯时间： 0830
=== 坐标信息处理 ===
逗号位置： 10
短横线位置： 3
标准化坐标： N3952.783,E11625.467
逗号位置： 9
纬度部分： N3952.783
经度部分： E11625.467
纬度方向： N
纬度数值： 3952.783
```

```
latitude = standard_coords[0:comma_position]
# 从开始到逗号之前
longitude = standard_coords[comma_position+1:]
# 从逗号之后到结束
print("纬度部分：", latitude)
print("经度部分：", longitude)
```

【案例分析】

问题4：字符串查找/切片 - 定位关键内容。

"ALPHA7|N39-52.783,E116-25.467|T0830|CONFIRMED"

```
分割后的信息块： ['ALPHA7', 'N39-52.783,E116-25.467', 'T0830', 'CONFIRMED']
信息块数量： 4
```

```
# 提取各个信息块
unit_code = message_parts[0]      # 单位代号
coordinates = message_parts[1]    # 坐标信息
time_stamp = message_parts[2]     # 时间戳
status = message_parts[3]         # 状态
```

```
# 进一步解析纬度：分离方向标识和数值
lat_direction = latitude[0]       # 第一个字符：N或S
lat_value = latitude[1:]          # 从第二个字符开始到最后
print("纬度方向：", lat_direction)
print("纬度数值：", lat_value)
```

```
单位名称： ALPHA
单位编号： 7
纯时间： 0830
=== 坐标信息处理 ===
逗号位置： 10
短横线位置： 3
标准化坐标： N3952.783,E11625.467
逗号位置： 9
纬度部分： N3952.783
经度部分： E11625.467
纬度方向： N
纬度数值： 3952.783
```

【案例分析】

问题6：字符串拼接 - 生成最终报告。

"ALPHA7|N39-52.783,E116-25.467|T0830|CONFIRMED"

```
========== 生成情报报告 ==========
【情报报告】 - 军事通信
单位：ALPHA-7
坐标：N3952.783,E11625.467
时间：0830
状态：CONFIRMED
==============================

【坐标详细分析】
纬度：N 3952.783
经度：E11625.467
格式化：N3952.783,E11625.467
```

```
print("========== 生成情报报告 ==========")
# 知识点：字符串拼接"+"
print("【情报报告】" + " - 军事通信")
# 方法一：使用 f-string（推荐）
print(f"单位：{unit_name}-{unit_number}\n
坐标：{standard_coords}")
print(f"时间：{pure_time}")
print(f"状态：{status}")
print("=" * 30)          #复制30次字符串"="
```

- Python提供了其他字符串的基本操作符，如表所示。

操作符	描述
x + y	连接两个字符串x与y
x * n 或 n * x	复制n次字符串x
x in s	如果x是s的子串，返回True，否则返回False

- 与操作符有关的实例如下：

```
>>> "Python语言" + "程序设计"
'Python语言程序设计'
>>> name = "Python语言" + "程序设计" + "基础篇"
>>> 'Python语言程序设计基础篇'
'Python语言程序设计基础篇'
>>> "GOAL!" * 3
'GOAL!GOAL!GOAL!'
>>> "Python语言" in name
True
>>> 'Y' in "Python语言"
False
```

输出方法一：f-string

f-string提供了一种简洁的格式化字符串的方式，它在形式上以f或F引领字符串，在字符串中使用“{变量名}”标明被替换的真实数据和其所在位置。

f('{变量名}') 或F('{变量名}') — **格式**

```
>>> date = '2024.11.11';computer = "PYTHON";cpu_usage = 10
>>> formatted_string = f"{date}: 计算机{computer}的cpu占用率为{cpu_usage}%"
>>> print(formatted_string)
2024.11.11: 计算机PYTHON的cpu占用率为10%
```

输出方法二：%格式化方式

字符串具有一种特殊的内置操作，它可以使用%进行格式化。

格式符	格式说明
%c	将对应的数据格式化为字符
%d	将对应的数据格式化为整数
%u	将对应的数据格式化为无符号整型
%o	将对应的数据格式化为无符号八进制数

```
>>> date = "2024.11.11";computer = "PYTHON";cpu_usage = 10
>>> formatted_string = "%s：计算机%s的cpu占用率为%d%%"%(date, computer, cpu_usage)
>>> print(formatted_string)
2024.11.11：计算机PYTHON的cpu占用率为10%
```

【案例分析】

问题6：字符串拼接 - 生成最终报告。

"ALPHA7|N39-52.783,E116-25.467|T0830|CONFIRMED"

```
========== 生成情报报告 ==========
【情报报告】 - 军事通信
单位：ALPHA-7
坐标：N3952.783,E11625.467
时间：0830
状态：CONFIRMED
==============================

【坐标详细分析】
纬度：N 3952.783
经度：E11625.467
格式化：N3952.783,E11625.467
```

```
print("========== 生成情报报告 ==========")
# 知识点：字符串拼接"+"
print("【情报报告】" + " - 军事通信")
# 方法二：使用%格式化输出，%s 是字符串的占位符
print("单位：%s-%s" % (unit_name, unit_number))
print("坐标：%s" % standard_coords)
print("时间：%s" % pure_time)
print("状态：%s" % status)
print("=" * 30)
```

输出方法三：format()格式化

- 字符串format()方法的基本使用格式是：

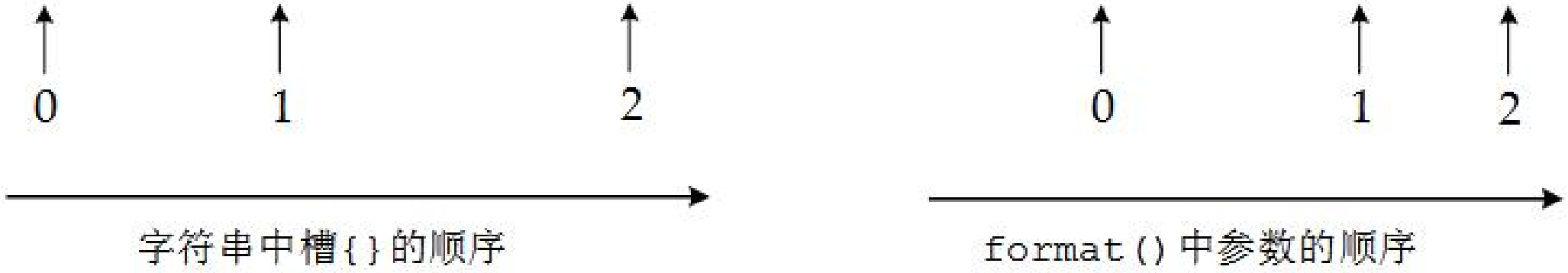
<模板字符串（包含槽{}）>.format(<逗号分隔的参数（变量名）>)

```
>>> date = "2024.11.11";computer = "PYTHON";cpu_usage = 10
2024.11.11：计算机PYTHON的cpu占用率为10%
```

输出方法三：format()格式化

- <模板字符串>由一系列的槽组成，用来控制修改字符串中嵌入值出现的位置，槽用大括号{}表示，如果大括号中没有序号，则按照出现顺序替换。

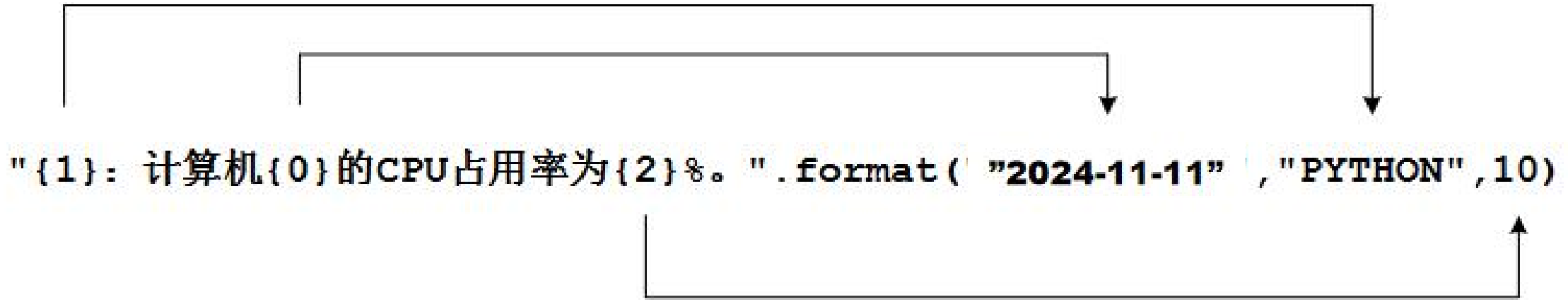
```
"{ }: 计算机{ }的CPU占用率为{ }%。".format( "2024-11-11" ,"PYTHON",10)
```




```
>>> "{}:计算机{}的CPU占用率为{}%,".format("2024-11-11","PYTHON",10)
'2024-11-11:计算机PYTHON的CPU占用率为10%,'
>>>
```

输出方法三：format()格式化

- 如果大括号中指定了使用参数的序号，按照序号对应参数替换，如图所示。调用format()方法后会返回一个新的字符串，format中的参数从0开始编号。



【案例分析】

问题6：字符串拼接 - 生成最终报告。

"ALPHA7|N39-52.783,E116-25.467|T0830|CONFIRMED"

```
========== 生成情报报告 ==========
【情报报告】 - 军事通信
单位：ALPHA-7
坐标：N3952.783,E11625.467
时间：0830
状态：CONFIRMED
==============================

【坐标详细分析】
纬度：N 3952.783
经度：E11625.467
格式化：N3952.783,E11625.467
```

```
print("========== 生成情报报告 ==========")
# 知识点：字符串拼接"+"
print("【情报报告】" + " - 军事通信")
# 方法三：使用format格式化输出，{}作为占位符
print("单位：{}-{}".format(unit_name, unit_number))
print("坐标：{}".format(standard_coords))
print("时间：{}".format(pure_time))
print("状态：{}".format(status))
print("=" * 30)
```

输出方法三：format()格式化

- format()方法中<模板字符串>的槽除了包括参数序号，还可以包括**格式控制信息**。

  此时，槽的内部样式如下：

  {<参数序号>:<格式控制标记>}

:	<填充>	<对齐>	<宽度>	,	<.精度>	<类型>
引导符号	用于填充的单个字符	< 左对齐 > 右对齐 ^ 居中对齐	槽的设定输出宽度	数字的千位分隔符 适用于整数和浮点数	浮点数小数部分的精度 或 字符串的最大输出长度	整数类型 b, c, d, o, x, X 浮点数类型 e, E, f, %

- **输出方法三：format()格式化**

- <格式控制标记>包括：<填充><对齐><宽度>,<.精度><类型>6个字段，这些字段都是可选的，可以组合使用。

<填充>、<对齐>和<宽度>是3个相关字段。

```
>>> s = "PYTHON"
>>> "{0:30}".format(s)
'PYTHON                        '
>>> "{0:>30}".format(s)
'                        PYTHON'
>>> "{0:*^30}".format(s)
'************PYTHON************'
>>> "{0:-^30}".format(s)
'------------PYTHON------------'
>>> "{0:3}".format(s)
'PYTHON'
```

- **输出方法三：format()格式化**

- <.精度>表示两个含义，由小数点（.）开头。对于浮点数，精度表示小数部分输出的有效位数。对于字符串，精度表示输出的最大长度。

```
>>> "{0:.2f}".format(12345.67890)
'12345.68'
>>> "{0:H^20.3f}".format(12345.67890)
'HHHHH12345.679HHHHHH'
>>> "{0:.4}".format("PYTHON")
'PYTH'
```

- **输出方法三：format()格式化**
- <类型>表示输出整数和浮点数类型的格式规则。

整数		浮点数	
b	输出整数的二进制方式	e	输出浮点数对应的小写字母e的指数形式
c	输出整数对应的Unicode字符	E	输出浮点数对应的大写字母E的指数形式
d	输出整数的十进制方式	f	输出浮点数的标准浮点形式
o	输出整数的八进制方式	%	输出浮点数的百分形式
x	输出整数的小写十六进制方式		
X	输出整数的大写十六进制方式		

【案例分析】

问题6：字符串拼接 - 生成最终报告。

"ALPHA7|N39-52.783,E116-25.467|T0830|CONFIRMED"

```
========== 生成情报报告 ==========
【情报报告】 - 军事通信
单位：ALPHA-7
坐标：N3952.783,E11625.467
时间：0830
状态：CONFIRMED
==============================

【坐标详细分析】
纬度：N 3952.783
经度：E11625.467
格式化：N3952.783,E11625.467
```

```
# 生成详细坐标分析
coord_analysis = f"""
【坐标详细分析】
纬度：{lat_direction} {lat_value}
经度：{longitude}
格式化：{standard_coords}
"""
print(coord_analysis)
```

另外两种输出方式？

【体育活动报名信息提取-1】

假设你有一个字符串，代表一个大学生报名参加体育活动的记录。记录的格式如下："姓名-性别-

例如："张三

请编写一个

```
# 输入记录字符串
record = input("请输入学生的体育活动报名信息（格式：姓名-性别-年龄-学院-活动名称）：")
# 提取姓名
name = record.split('-')[0]
print("姓名:", name)
activity = record.split('-')[-1]# 提取活动名称
print("活动名称:", activity)
```

输入：接受用户输入的一条记录字符串。

输出：

提取并打印出该学生的姓名（第一个"-"之前的部分）。

提取并打印出该学生报名参加的活动名称（最后一个"-"之后的部分）。

字符串操作方法

【体育活动报名信息提取-2】

假设你有一个字符串，代表一个大学生报名参加体育活动的记录。记录的格式如下：

"姓名-性别-年龄-学院-活动名称"

请编写一个Python程序，完成

输出：

学生信息：

姓名：{姓名}

性别：{性别}

年龄：{年龄}

学院：{学院}

活动名称：{活动名称}

```
record = input("请输入学生的体育活动报名信息（格式：姓名-性别-年龄-学院-活动名称）：")
# 提取姓名，假设姓名长度为2个字符
name = record[0:2]
# 提取性别，假设长度为1个字符
sex = record[3:4]
# 提取年龄，假设年龄长度为2个字符
age = record[5:7]
# 提取学院，假设学院长度为4个字符
college = record[-5:-3]
# 提取活动名称
activity = record[-2:]
# 使用format方法格式化输出学生信息
print("学生信息：")
print("姓名：{}".format(name))
print("性别：{}".format(sex))
print("年龄：{}".format(age))
print("学院：{}".format(college))
print("活动名称：{}".format(activity))
```

其中 {姓名}、{年龄}、{学院} 和 {活动名称} 是通过format方法替换的变量。

02

第三方库

2. 第三方库概念

Python计算生态涵盖网络爬虫、数据分析、文本处理、数据可视化、图形用户界面、机器学习、Web开发、网络应用开发、游戏开发、虚拟现实、图形艺术等多个领域，为各个领域的Python使用者提供了极大便利。

第三方库

- Python拥有十分丰富的第三方库

……

math库
利用基本功能实现：
开根号、sin、cos…

……

Python
只提供+-*/等基本功能

第1关：变量与赋值

学习内容　参考答案　记录

第二题

钟形高斯函数如下所示，请在指定位置编写程序，计算不同参数下的结果。

$$f(x)=\frac{1}{\sqrt{2\pi}s}e^{\left[-\frac{1}{2}\left(\frac{x-m}{s}\right)^2\right]}$$

```
from math import pi,sqrt,pow,e

fx=(1/sqrt(2*pi*s))*pow(e,-((1/2))*pow((x-m)/s,2))
```

第三方库的使用

库是Python中常常提及的概念，但事实上Python中的库只是一种对特定功能集合的统一说法而非严格定义。Python库的具体表现形式为模块（Module）和包（Package）。

导入模块：

方式一：import 模块名

方式二：from ... import ...

```
>>> import math
>>> result = math.sqrt(9)
>>> print(result)
3.0
```

- 引用函数库的方法如下：
  - from <库名> import <函数名,函数名,...,函数名>
  - from <库名> import * #其中，*是通配符，表示所有函数
- 此时，调用该库函数时不需要使用库名，直接使用如下格式：
  - <函数名>(<函数参数>)

time模块

time模块包含的内置函数

time是最基础的时间处理库，该库本质上是一个模块，该库中定义了localtime()、asctime()、strftime()和一些用于实现时间格式转换的函数。

1.获取当前时间

从返回浮点数的时间戳方式向时间元组转换，只要将浮点数传递给如localtime()之类的函数即可。

【示例】

```
>>>import time
>>>current_time1=time.localtime()
>>>print("本地时间为：",current_time1)
```

本地时间为：time.struct_time(tm_year=2025, tm_mon=11, tm_mday=10, tm_hour=16, tm_min=26, tm_sec=11, tm_wday=0, tm_yday=314, tm_isdst=0)

time.localtime() 返回的是一个 time.struct_time 对象，包含以下字段：

tm_year: 年份（如：2024）
tm_mon: 月份（1-12）
tm_mday: 一个月中的第几天（1-31）
tm_hour: 小时（0-23）
tm_min: 分钟（0-59）
tm_sec: 秒（0-59）
tm_wday: 一周中的第几天（0-6，0表示星期一）
tm_yday: 一年中的第几天（1-366）
tm_isdst: 夏令时标志（0表示否，1表示是，-1表示未知）

time模块

2. 获取格式化的时间

非常简单的获取可读的格式化时间的函数是asctime()。

例如：

```
>>>import time
>>>current_time2 =time.asctime()
>>>print("本地时间为：",current_time2)
```

```
本地时间：  Mon Nov 10 16:32:52 2025
```

time模块

3. 格式化日期数据

可以使用time模块的strftime()函数来格式化日期数据，其

```
time.strftime(fmt[,tupletimet])
```

把一个代表时间的元组或者struct_time元组（例如由time.localtime()和time.gmtime()返回）转化为格式化的时间字符串。如果tupletime未指定，将传入time.localtime()，如果元组中任意一个元素取值超出允许范围，将会抛出ValueError异常。

time模块

【实例】演示格式化日期数据

```
import time
#格式化成2025-11-11-17:10:32形式
  print(time.strftime("%Y-%m-%d-%H:%M:%S",time.localtime()))
#格式化成Thu-Apr-02-17:11:232020形式
  print(time.strftime("%a-%b-%d-%H:%M:%S:%Y",time.localtime()))
#将格式字符串转换为时间戳
  t="Mon-Nov-11-01:14:43:2025"
  print(time.mktime(time.strptime(t,"%a-%b-%d-%H:%M:%S:%Y")))
```

实例的运行结果如下。

```
2025-11-11-01:14:43
Mon-Nov-11-01:14:43:2025
1731258883.0
```

格式符号说明

格式符号	说明	示例
%Y	4位数年份	2024
%y	2位数年份	24
%m	2位数月份	11
%b	月份缩写	Nov
%B	月份全名	November
%d	2位数日期	05
%a	星期缩写	Tue
%A	星期全名	Tuesday
%H	24小时制小时	14
%I	12小时制小时	02
%M	分钟	30
%S	秒	25
%p	AM/PM	PM

作业

- 处理通信数据："CHARLIE5#N12-34.567,E98-76.543#T2100#PENDING"，使用今天学习的所有方法，生成完整的情报报告。
- 使用time模块编写一个程序，获取当前时间，并格式化输出为“2025年11月11日 8时8分8秒”的形式。

总结

- 用字符串方法清洗、提取数据，关乎情报的准确性。
- 用f-string/%/format格式化信息，关乎指令的清晰度。

这正如我们的人生：字符串操作好比我们日常的修行，需要不断“清洗”掉坏习惯，“格式化”出好品格。而第三方库则象征着我们身边的良师益友、书籍知识，善于借助这些外部力量，我们才能更快地成长。

技术的价值在于应用。 希望你们不仅能运用这些知识写出优雅的代码，更能用它们去解决实际问题，承担起新时代学子的科技报国使命。

下课并不代表思考的终止
期待我们下次的思想碰撞