



首都师范大学

為學為師 求實求新

高级程序设计

---Python与深度学习

2. Python基础语法

李冰

副研究员

交叉科学研究院



回顾

- 课程内容
 - Python基本语法
 - 深度学习基础
- Python 开发环境安装
 - Windows 下 python下载安装
 - 包管理器: pip, conda
 - 集成开发环境: Jupyter Notebook, VS code, Pycharm (教育版)

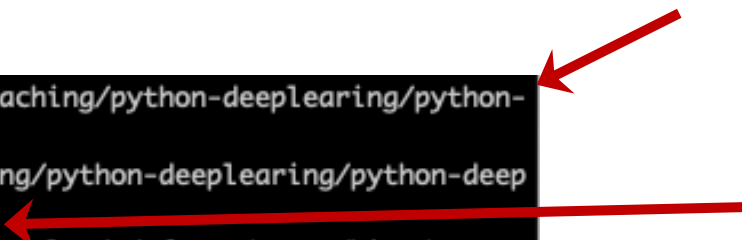
课程内容

- Python 基础语法
 - 注释
 - 代码格式
 - 行与缩进
 - 代码习惯
- 变量
 - 变量命名
 - 变量类型
 - 数字
 - 字符串

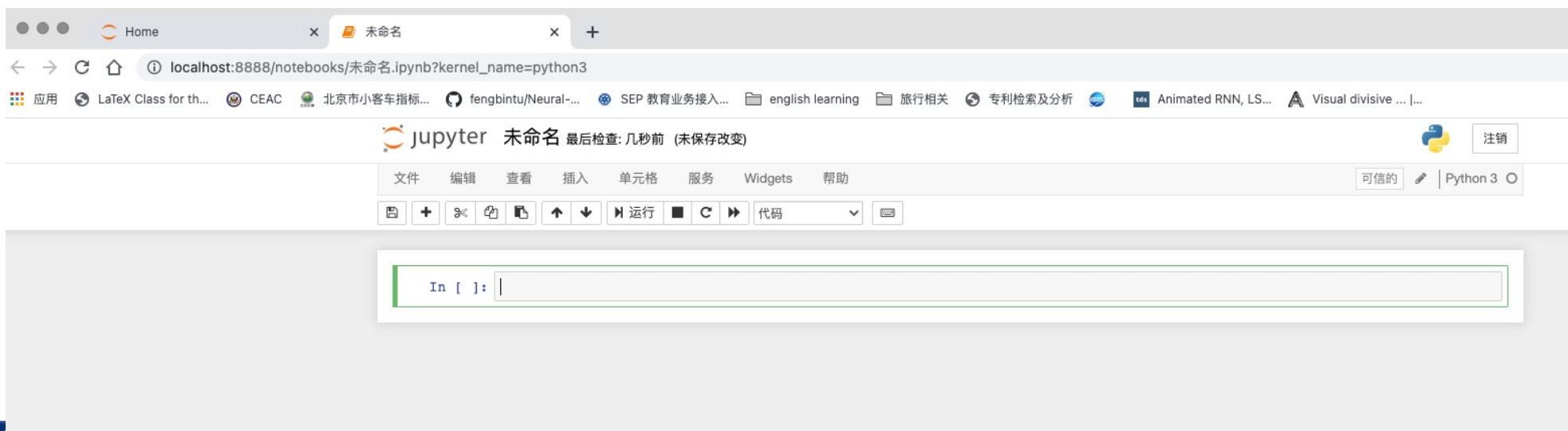
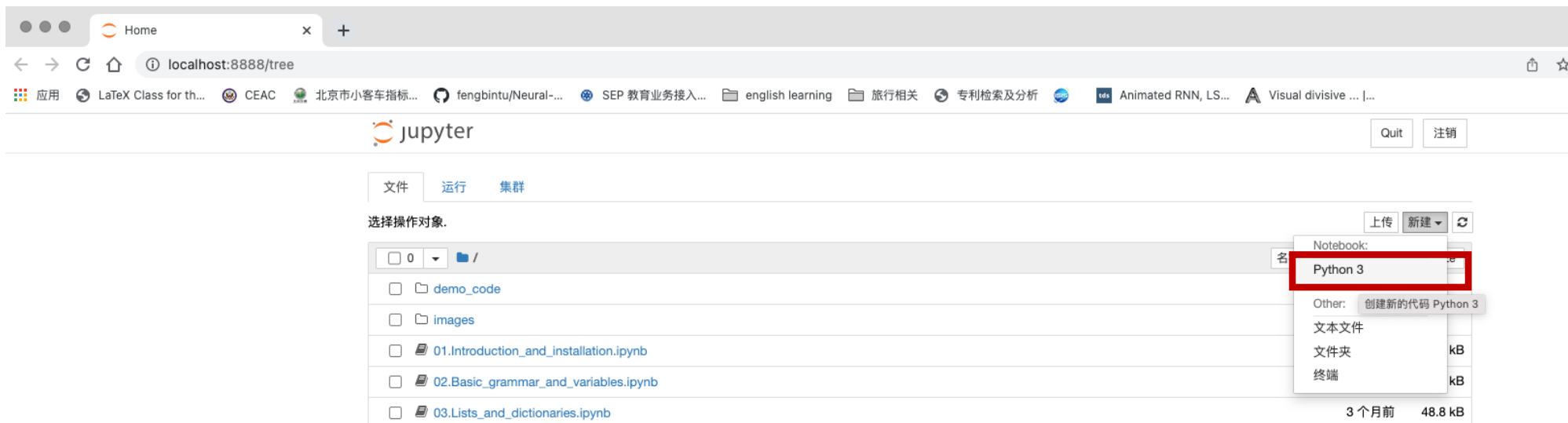
启动Jupyter Notebook

```
(base) [libin@Bing-Pro ~]$cd OfflineDocument/Teaching/python-deeplearning/python-deep-learning-master/
(base) [libin@Bing-Pro ~/OfflineDocument/Teaching/python-deeplearning/python-deep-learning-master]$jupyter notebook
[I 22:07:24.020 NotebookApp] JupyterLab extension loaded from /Users/bing/anaconda3/lib/python3.7/site-packages/jupyterlab
[I 22:07:24.020 NotebookApp] JupyterLab application directory is /Users/bing/anaconda3/share/jupyter/lab
[I 22:07:24.025 NotebookApp] 启动notebooks 在本地路径: /Users/bing/OfflineDocument/Teaching/python-deeplearning/python-deep-learning-master
[I 22:07:24.025 NotebookApp] 本程序运行在: http://localhost:8888/?token=201b9e8b02069bcdcd426079e9944486ae1a23c54b3caa61
[I 22:07:24.025 NotebookApp] 使用 control-c停止此服务器并关闭所有内核(两次跳过确认).
[C 22:07:24.045 NotebookApp]

To access the notebook, open this file in a browser:
    file:///Users/bing/Library/Jupyter/runtime/nbserver-37062-open.html
Or copy and paste one of these URLs:
    http://localhost:8888/?token=201b9e8b02069bcdcd426079e9944486ae1a23c54b3caa61
```



启动Jupyter Notebook



Python 程序

- Python 程序可以分解为模块、语句、表达式以及对象：
 - 程序由**模块**构成
 - 模块包含**语句**
 - 语句包含**表达式**
 - 表达式建立并处理**对象 (变量、类、实例)**

Python基础语法—注释

- Python 中单行注释以 # 开头

```
# 第一个注释  
print('Hello, Python!')      # 第二个注释
```

Hello, Python!

- 多行注释可以用多个 # 号，或者使用三个单引号 ''' 或三个双引号 """

```
# 第一个注释  
# 第二个注释
```

```
'''  
第三个注释  
第四个注释  
'''
```

```
"""  
第五个注释  
第六个注释  
"""
```

```
print('Hello, Python!')
```

Hello, Python!

Python基础语法—行与缩进

- Python 最具特色的就是使用缩进来表示代码块，不需要使用大括号 {}。
 - 代码块内的语句会垂直对齐，块会在碰到缩进量较少的行或文件末尾时结束，而更深层的嵌套块比所在块进一步向右缩进。
- 缩进的空格数是可变的（默认为四个空格），同一个代码块的语句必须包含相同的缩进空格数。
 - 多数IDE会在输入代码时自动缩进来符合 Python 规范。
- 避免混合使用Tab键和空格

```
x = 1
if x:
    y = 2
    if y:
        print('block2')
    print('block1')
print('block0')
```

block2
block1
block0

```
if False:
    print('True')
else:
    print('Something')
    #print("False")    # 缩进不一致，会导致运行错误
```

Something

代码习惯

• 空行

- 代码块之间用空行分隔，表示一段新的代码的开始。
- 空行与代码缩进不同，书写时不插入空行，Python解释器运行也不会出错。
- 空行的作用在于分隔两段不同功能或含义的代码，便于日后代码的维护或重构。

• 同一行显示多条语句

- Python允许在同一行中编写多条非复合语句（语句内未嵌套其它语句），语句之间使用分号(;)分割。

```
import sys; x = 'run'; print(x)
```

没有可读性

import 与 from...import

- 在 Python 中用 import 或者 from...import 来导入相应的模块。
 - 将整个模块(somemodule)导入, 格式为: `import somemodule`
 - 从某个模块中导入某个函数, 格式为: `from somemodule import somefunction`
 - 从某个模块中导入多个函数, 格式为: `from somemodule import firstfunc, secondfunc, thirdfunc`
 - 将某个模块中的全部函数导入, 格式为: `from somemodule import *`

```
: import sys
print('Command line arguments: ')
for i in sys.argv:
    print(i)
print('System path: ', sys.path)
```

import 与 from...import

- 在 Python 中用 import 或者 from...import 来导入相应的模块。
 - 将整个模块(somemodule)导入, 格式为: `import somemodule`
 - 从某个模块中导入某个函数, 格式为: `from somemodule import somefunction`
 - 从某个模块中导入多个函数, 格式为: `from somemodule import firstfunc, secondfunc, thirdfunc`
 - 将某个模块中的全部函数导入, 格式为: `from somemodule import *`

```
: import sys
print('Command line arguments: ')
for i in sys.argv:
    print(i)
print('System path: ', sys.path)
```

Number of arguments: 3 arguments.

Arguments List: ['/Users/bing/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py', '-f', '/Users/bing/Library/Jupyter/runtime/kernel-6d4ec0f3-da8d-4986-be2b-82372799dca6.json']

Command line arguments:

/Users/bing/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py

-f

/Users/bing/Library/Jupyter/runtime/kernel-6d4ec0f3-da8d-4986-be2b-82372799dca6.json

变量命名规则

- 关键字

- 这些保留字不能用作常量或变量，或任何其他标识符名称。

```
import keyword  
print(len(keyword.kwlist), keyword.kwlist)
```

```
35 ['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'el  
if', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not',  
'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

变量命名规则

• 关键字

- 这些保留字不能用作常量或变量，或任何其他标识符名称。

```
import keyword
print(len(keyword.kwlist), keyword.kwlist)
```

```
35 ['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'el
if', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not',
'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

• 变量命名规则

- 第一个字符必须是字母表中字母或下划线 _。
- 标识符的其他部分由字母、数字和下划线组成。
 - spam、_spam、Spam_1 都是合法的变量名，但是 1_Spam、spam\$ 以及 @#! 则不是合法变量名。
- 标识符对大小写敏感
 - 例如，spam 和 SPAM 是两个不同的变量名
- 禁止使用关键字
 - 定义的变量名不能和 Python 中关键字相同。

变量类型

- 下表是 Python 的核心对象类型和生成这些对象的表达式。

| 对象类型 | 英文名 | 示例 |
|--------|--------------------|--|
| 数字 | Numbers | 1234, 3.1415, 3+4j, 0b111, Decimal(), Fraction() |
| 字符串 | Strings | 'spam', "guido's", b'a\x01c', u'sp\xc4m' |
| 列表 | Lists | [1, [2, 'three'], 4.5], list(range(10)) |
| 字典 | Dictionaries | {'food': 'spam', 'taste': 'yum'}, dict(hours=10) |
| 元组 | Tuples | (1, 'spam', 4, 'U'), tuple('spam'), namedtuple |
| 文件 | Files | open('eggs.txt'), open(r'C:\ham.bin', 'wb') |
| 集合 | Sets | set('abc'), {'a', 'b', 'c'} |
| 其他类型 | Other core types | None |
| 编程单元类型 | Program unit types | 函数、模块、类 |

- Python 中每个变量在使用前都必须赋值，变量赋值以后该变量才会被创建。
- 等号运算符（=）用来给变量赋值
 - 左边是一个变量名，右边是存储在变量中的值。

变量类型

- 变量赋值和声明

```
counter = 100          # 整型变量
miles   = 1000.0        # 浮点型变量
name    = 'python'      # 字符串

print(type(counter))
print(type(miles))
print(type(name))
```

变量类型

- 变量赋值和声明

```
counter = 100          # 整型变量
miles   = 1000.0       # 浮点型变量
name    = 'python'     # 字符串

print(type(counter))
print(type(miles))
print(type(name))

<class 'int'>
<class 'float'>
<class 'str'>
```


变量类型

- 变量赋值和声明

```
# Python可以同时为多个变量赋值
```

```
a, b, c, d = 20, 5.5, True, 4+3j
```

```
print(type(a), type(b), type(c), type(d))
```

```
<class 'int'> <class 'float'> <class 'bool'> <class 'complex'>
```

变量类型

- **不可变类型/对象（数字、字符串、元组）**
 - 不可变类型的对象不支持原地修改，我们可以创建新的对象来存储表达式的运行结果。
- **可变类型/对象（列表、字典）**
 - 可变的类型总是可以通过操作原地修改，而不用创建新的对象。

| | | |
|-----|--------|--|
| 数字 | Number | 1234, 3.1415, 3+4j, 0b111, Decimal(), Fraction() |
| 字符串 | String | 'spam', "guido's", b'a\x01c', u'sp\xc4m' |
| 元组 | Tuple | (1, 'spam', 4, 'U'), tuple('spam'), namedtuple |

| | | |
|----|------------|--|
| 列表 | List | [1, [2, 'three'], 4.5], list(range(10)) |
| 字典 | Dictionary | {'food': 'spam', 'taste': 'yum'}, dict(hours=10) |

课程内容

- Python 基础语法
 - 注释
 - 代码格式
 - 行与缩进
 - 代码习惯
- 变量
 - 变量命名
 - 变量类型
 - 数字
 - 字符串

数字

- Python 中数字有多种类型：整数、浮点数、布尔类型、分数类型、复数类型和二进制、八进制、十六进制记数。
 - int (整数), 如 1, 只有一种整数类型 int, 表示为长整型, 没有 Python2 中的 Long
 - float (浮点数), 如 1.23、3E-2
 - bool (布尔类型), 如 True, False
 - fraction (分数类型), 如 Fraction(1, 4)
 - complex (复数类型), 如 1 + 2j、1.1 + 2.2j
 - binary, octal, hex (二进制、八进制和十六进制)

- 操作符

| 除法 | 整除 | 取余 | n次方 |
|--------|---------|--------|--------|
| / | // | % | ** |
| 11 / 3 | 11 // 3 | 11 % 3 | 2 ** 4 |

```
11 / 3
3.6666666666666665
```

```
11 // 3
3
```

```
11 % 3
2
```

内置数学模块

- 除了核心对象类型和表达式之外，和 Python 一起分发的还有一些常用的数学模块，模块只不过是我們导入以供使用的一些额外工具包。
 - math, numpy, pandas, scipy等
- **math** 模块包括更高级的数学工具，如数学常量和数学函数

```
import math
math.pi, math.e           # Common constants
```

```
(3.141592653589793, 2.718281828459045)
```

```
math.sqrt(85)             # Square root
```

```
9.219544457292887
```

```
math.sin(2 * math.pi / 180) # Sine, tangent, cosine
```

```
0.03489949670250097
```

内置数学模块

- math 模块中还包含了trunc截断、floor、ceil方法

```
math.floor(2.567), math.floor(-2.567)    # Floor (next-lower integer)  
(2, -3)
```

```
math.ceil(2.567), math.ceil(-2.567)     # Ceil (next-upper integer)  
(3, -2)
```

```
math.trunc(2.567), math.trunc(-2.567)    # Truncate (drop decimal digits)  
(2, -2)
```

内置数学模块

- math 模块中还包含了trunc截断、floor、ceil方法
- Python用内置函数 int() 和 round() 进行取整操作，前者会直接省略小数部分，而后者进行四舍五入：

```
: int(2.567), int(-2.567)                                # Truncate (integer conversion)
```

```
: (2, -2)
```

```
: round(2.567), round(2.467), round(2.567, 2)          # Round
```

```
: (3, 2, 2.57)
```

```
: '%.1f' % 2.567, '{0:.2f}'.format(2.567)                # Round for display
```

```
: ('2.6', '2.57')
```

内置数学模块

- random 模块可以作为随机数字的生成器和随机选择器。

```
import random
random.random()           # Return the next random floating point number in the range [0.0, 1.0)
0.2864884203924346
```

```
random.randint(1, 10)     # Return a random integer N such that a <= N <= b
4
```

```
random.choice([1, 2, 3, 4]) # Return a random element from the non-empty sequence seq
1
```


布尔类型

- Python 拥有明确的布尔型数据类型，叫做 bool，其值为 True 和 False，并且其值 True 和 False 是预先定义的内置的变量名。
 - 在内部，变量名 True 和 False 是 bool 的实例，实际上仅仅是内置的整数类型 int 的子类（以面向对象的观点来看）。
 - True 和 False 的行为和整数 1 和 0 是一样的，除了它们有特定的显示逻辑：它们是作为关键字 True 和 False 显示的，而不是数字 1 和 0。

```
type(True)
```

```
bool
```

```
True == 1          # ==操作符测试两个对象是否有相同的值
```

```
True
```

```
True + 4           # 你不可能在真正的 Python 代码中遇到这样的表达式
```

```
5
```

```
True is 1          # is操作符检查两个对象的同一性
```

布尔类型

- Python 拥有明确的布尔型数据类型，叫做 bool，其值为 True 和 False，并且其值 True 和 False 是预先定义的内置的变量名。
 - 在内部，变量名 True 和 False 是 bool 的实例，实际上仅仅是内置的整数类型 int 的子类（以面向对象的观点来看）。
 - True 和 False 的行为和整数 1 和 0 是一样的，除了它们有特定的显示逻辑：它们是作为关键字 True 和 False 显示的，而不是数字 1 和 0。

```
type(True)
```

```
bool
```

```
True == 1          # ==操作符测试两个对象是否有相同的值
```

```
True
```

```
True + 4          # 你不可能在真正的 Python 代码中遇到这样的表达式
```

```
5
```

```
True is 1         # is操作符检查两个对象的同一性
```

!False!

复数类型

- 复数表示为两个浮点数（实部和虚部）并接在虚部增加了j或J的后缀。我们能够把非零实部的复数写成由 + 连接起来的两部分。
 - 例如，一个复数的实部为2，并且虚部为-3可以写成 $2 + -3j$ 。
- 下面是一些复数运算的例子。

```
In [38]: 1j * 1j
```

```
Out[38]: (-1+0j)
```

```
In [39]: 2 + 1j * 3
```

```
Out[39]: (2+3j)
```

```
In [40]: (2 + 1j) * 3
```

```
Out[40]: (6+3j)
```

二进制、八进制和十六进制记数

- Python 整数能够以二进制、八进制和十六进制记数法来编写，作为一般的十进制记数法的补充
 - 以 **0b**, **0o** 和 **0x** 开头。

```
In [41]: 0b1, 0b10000, 0b11111111    # Binary literals
```

```
Out[41]: (1, 16, 255)
```

```
In [42]: 0o1, 0o20, 0o377           # Octal literals
```

```
Out[42]: (1, 16, 255)
```

```
In [43]: 0x01, 0x10, 0xff           # Hex literals
```

```
Out[43]: (1, 16, 255)
```

二进制、八进制和十六进制记数

- Python 提供了内置的函数，可以将整数转换为其他进制的数字字符串。

```
In [44]: bin(64), oct(64), hex(64)
```

```
Out[44]: ('0b1000000', '0o100', '0x40')
```

二进制、八进制和十六进制记数

- Python 提供了内置的函数，可以将整数转换为其他进制的数字字符串。

```
In [44]: bin(64), oct(64), hex(64)
```

```
Out[44]: ('0b1000000', '0o100', '0x40')
```

- Python 内置的 `int()` 函数可以将一个数字的字符串变换为一个整数，并可以通过第二个参数来确定变换后的数字的进制

```
In [45]: int('64'), int('1000000', 2), int('100', 8), int('40', 16)
```

```
Out[45]: (64, 64, 64, 64)
```

```
In [46]: int('0x40', 16), int('0b1000000', 2)
```

```
Out[46]: (64, 64)
```

二进制、八进制和十六进制记数

- Python 提供了内置的函数，可以将整数转换为其他进制的数字字符串。

```
In [44]: bin(64), oct(64), hex(64)
Out[44]: ('0b1000000', '0o100', '0x40')
```

- Python 内置的 int() 函数可以将一个数字的字符串变换为一个整数，并可以通过第二个参数来确定变换后的数字的进制

```
In [45]: int('64'), int('1000000', 2), int('100', 8), int('40', 16)
Out[45]: (64, 64, 64, 64)
```

```
In [46]: int('0x40', 16), int('0b1000000', 2)
Out[46]: (64, 64)
```

- 使用字符串格式化方法将一个整数转换成二进制、八进制或十六进制数的字符串。

```
In [47]: '{0:b}, {1:o}, {2:x}'.format(64, 64, 64)
Out[47]: '1000000, 100, 40'
```

位运算

- 除了一般的数学运算，Python 也支持位运算

```
In [48]: x = 1  
x << 2      # Shift left 2 bits: 0100
```

```
Out[48]: 4
```

```
In [49]: x | 2      # OR
```

```
Out[49]: 3
```

```
In [50]: x & 1      # AND
```

```
Out[50]: 1
```


位运算

```
In [24]: x = 0b0001      # Binary literals  
x << 2
```

```
Out[24]: 4
```

```
In [52]: bin(x << 2)
```

```
Out[52]: '0b100'
```

```
In [53]: bin(x | 0b010) # Bitwise OR
```

```
Out[53]: '0b11'
```

```
In [54]: bin(x & 0b1)   # Bitwise AND
```

```
Out[54]: '0b1'
```

位运算

•十六进制与逻辑运算

```
In [55]: X = 0xFF          # Hex literals  
         bin(X)
```

```
Out[55]: '0b11111111'
```

```
In [56]: X ^ 0b10101010    # Bitwise XOR
```

```
Out[56]: 85
```

```
In [57]: bin(X ^ 0b10101010)
```

```
Out[57]: '0b1010101'
```

分数

- 从 Python 2.6和 Python 3.0 开始引入了分数，它明确保留一个分子和一个分母，从而避免了浮点数学的某些不精确性和局限性。

```
In [30]: from fractions import Fraction  
x = Fraction(1, 3)  
print(x)  
print(type(x))
```

```
1/3  
<class 'fractions.Fraction'>
```

```
In [31]: y = Fraction(4, 6)           # Simplified to 2, 3  
print(y)
```

```
2/3
```

分数类型

- 一旦创建了分数，它可以像平常一样用于数学表达式中：

```
In [19]: x + y, x - y, x * y
```

```
Out[19]: (Fraction(1, 1), Fraction(-1, 3), Fraction(2, 9))
```

```
In [21]: print(x + 2)
          print(x + 2.0)
          print(x + Fraction(4, 3))
```

```
7/3
```

```
2.3333333333333335
```

```
5/3
```

- 分数对象也可以从浮点数字符串来创建，这和小数很相似：

```
In [34]: Fraction('.25')
```

```
Out[34]: Fraction(1, 4)
```

```
In [35]: Fraction('.25') + Fraction('1.25')
```

```
Out[35]: Fraction(3, 2)
```

分数类型

- 对于那些用内存中给定的有限位数无法精确表示的值，浮点数的局限尤为明显。分数却能提供得到精确结果的方式，虽然要付出一些速度的代价。

```
In [22]: 0.1 + 0.1 + 0.1 - 0.3
```

```
Out[22]: 5.551115123125783e-17
```

```
In [23]: Fraction(1, 10) + Fraction(1, 10) + Fraction(1, 10) - Fraction(3, 10)
```

```
Out[23]: Fraction(0, 1)
```

回顾

- Python 基础语法
 - 注释
 - 代码格式
 - 行与缩进
 - 代码习惯
- 变量
 - 变量命名
 - 变量类型
 - 数字
 - 字符串

课程内容

- Python 基础语法
 - 注释
 - 代码格式
 - 行与缩进
 - 代码习惯
- 变量
 - 变量命名
 - 变量类型
 - 数字
 - 字符串

字符串(String)

- 一个有序的字符的集合，用来存储和表现基于文本的信息。
- 从功能的角度来看，字符串可以用来表示能够像文本那样编辑的任何信息：符号和词语、载入到内存中的文本文件的内容、Internet网址和Python程序等。
- 有多种方法编写 Python 中的字符串常量：
 - 单引号: 'spa'm'
 - 双引号: "spa'm"
 - 三引号: "...spam...", "...spam..."

```
In [58]: paragraph = """This is a paragraph,  
consist of multiple lines."""  
print(paragraph)
```

```
This is a paragraph,  
consist of multiple lines.
```


单双引号字符串是一样

- 可以在一个双引号字符串所包含的字符串中嵌入一个单引号字符

```
In [68]: 'knight"s', "knight's"
```

```
Out[68]: ('knight"s', "knight's")
```

```
In [69]: 'knight\'s', "knight\"s" # 使用转义字符
```

```
Out[69]: ("knight's", 'knight"s')
```

- Python 自动在任意的表达式中合并相邻的字符串常量（可以简单使用+操作符来明确表示这是一个合并操作）

```
In [70]: title = "Meaning " 'of' " Life" # Implicit concatenation  
title
```

```
Out[70]: 'Meaning of Life'
```

字符串

- 使用 for 语句在一个字符串中进行循环迭代。

```
: myjob = 'hacker'  
  for c in myjob:  
      print(c, end=' ')    # Step through items  
  
h a c k e r
```

for 循环指派了一个变量去获取一个字符串中的元素，并为每个元素执行一段语句。这里，变量 c 成为了在这个字符串中步进的指针

字符串

- 字符串可以通过 + 操作符进行合并，并且可以通过 * 操作符进行重复：

```
In [59]: len('abc')           # Length: number of items
```

```
Out[59]: 3
```

```
In [60]: 'abc' + 'def'       # Concatenation: a new string
```

```
Out[60]: 'abcdef'
```

```
In [61]: 'Ni!' * 4           # Repetition: like "Ni!" + "Ni!" + ...
```

```
Out[61]: 'Ni!Ni!Ni!Ni!'
```

```
In [62]: print('-' * 80)
```

字符串

- 使用 in 表达式操作符对字符和子字符串进行包含关系的测试

```
In [64]: 'k' in myjob
```

```
Out[64]: True
```

```
In [65]: 'z' in myjob
```

```
Out[65]: False
```

```
In [66]: 'spam' in 'abcspamdef'
```

```
Out[66]: True
```

转义序列 (escape sequences)

- 转义序列让我们能够在字符串中嵌入不容易通过键盘输入的字节。
- 字符'\'以及在其后边的一个或多个字符，在最终的字符串对象中会被一个单个字符所代替。

| 转义 | 意义 |
|----|-------|
| \b | 倒退 |
| \f | 换页 |
| \n | 换行 |
| \r | 返回 |
| \t | 水平制表符 |
| \v | 垂直制表符 |

In [51]:

```
word = 'sentence'  
print('\t', word) #tab制表符  
print('\n', word) #换行符  
print('\r', word) #返回  
print(word, '\r', word) #返回
```

sentence

sentence
sentence
sentence

转义序列 (escape sequences)

- 为了清楚地了解这个字符串中到底有多少个字节，可以使用内置的 `len()` 函数。它会返回一个字符串中到底有多少字节，无论它是如何显示的。

```
s = 'a\nb\tc'  
print(s)
```

```
a  
b      c
```

```
In [73]: len(s)
```

```
Out[73]: 5
```

raw字符串抑制转义

- 为了引入转义字符而使用适应的反斜杠的处理会带来一些麻烦。例如，像下面这样使用文件名参数去尝试打开一个文件：

```
: # myfile = open('C:\new\text.dat', 'w')
```

本意是打开一个在 `C:\new` 目录下名为 `text.dat` 的文件。

问题是这里有 `"\n"`，它会被识别为一个换行字符，`"\t"` 会被一个制表符所替代。计算机尝试打开一个名为 `C:(换行)ew(制表符)ext.dat` 的文件。

raw字符串抑制转义

- 为了引入转义字符而使用适应的反斜杠的处理会带来一些麻烦。例如，像下面这样使用文件名参数去尝试打开一个文件：

```
: # myfile = open('C:\new\text.dat', 'w')
```

本意是打开一个在 `C:\new` 目录下名为 `text.dat` 的文件。

问题是这里有 `"\n"`，它会被识别为一个换行字符，`"\t"` 会被一个制表符所替代。计算机将尝试打开一个名为 `C:(换行)ew(制表符)ext.dat` 的文件。

```
# myfile = open(r'C:\new\text.dat', 'w')  
# myfile = open('C:\\new\\text.dat', 'w')
```

如果字母 `r`（小写或大写）出现在字符串引号的前面，它将关闭转义机制，Python 将反斜杠作为常量来保持。

三重引号编写多行字符串块

- 以三重引号开始（单引号和双引号都可以），并紧跟任意行数的文本，并且以开始时的同样的三重引号结尾。

```
In [52]: mantra = '''Always look  
          on the bright  
          side of life.'''  
mantra
```

```
Out[52]: 'Always look\n on the bright\nside of life.'
```

Python 将所有在三重引号内的文本收集到一个单独的多行字符串中，并在代码换行处嵌入了换行字符(\n)。

三重引号编写多行字符串块

- 以三重引号开始（单引号和双引号都可以），并紧跟任意行数的文本，并且以开始时的同样的三重引号结尾。

```
In [52]: mantra = '''Always look  
         on the bright  
         side of life.'''  
mantra
```

```
Out[52]: 'Always look\n on the bright\nside of life.'
```

Python 将所有在三重引号内的文本收集到一个单独的多行字符串中，并在代码换行处嵌入了换行字符(\n)。

- 要查看带有换行解释的字符串，需要使用 `print()` 函数。

```
: print(mantra)  
  
Always look  
  on the bright  
side of life.
```

三重引号编写多行字符串块

- 以三重引号开始（单引号和双引号都可以），并紧跟任意行数的文本，并且以开始时的同样的三重引号结尾。

```
In [52]: mantra = '''Always look  
         on the bright  
         side of life.'''  
mantra
```

```
Out[52]: 'Always look\n on the bright\nside of life.'
```

Python 将所有在三重引号内的文本收集到一个单独的多行字符串中，并在代码换行处嵌入了换行字符(\n)。

- 要查看带有换行解释的字符串，需要使用 `print()` 函数。

```
: print(mantra)  
  
Always look  
  on the bright  
  side of life.
```

三重引号字符串经常在开发过程中作为一种注释手段。

课程内容

- Python 基础语法
 - 注释
 - 代码格式
 - 行与缩进
 - 代码习惯
- 变量
 - 变量命名
 - 变量类型
 - 数字
 - 字符串

字符串—索引和分片

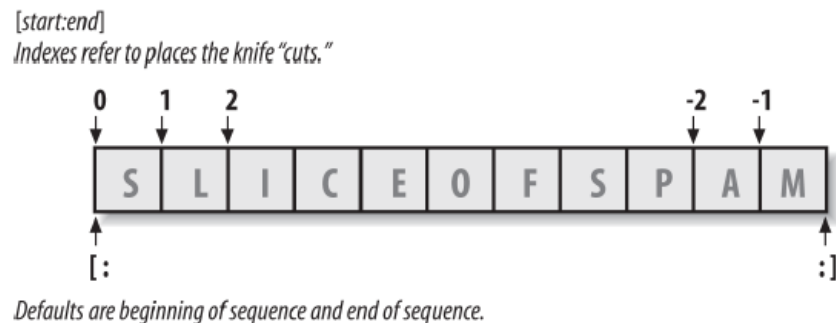
- 字符串定义为字符的有序集合，所以我们能够通过其位置获得它们的元素。在 Python 中，字符串中的字符是通过索引提取的。
- Python 偏移量是从 0 开始的，并比字符串的长度小1。
- Python 还支持在字符串中使用负偏移从序列中获取元素。
 - 从技术来讲，一个负偏移与字符串的长度相加后得到这个字符串的正的偏移值。可以将负偏移看做是从结束处反向计数。

字符串—索引和分片

•索引

- 索引（S[i]）获取特定偏移的元素：
- 第一个元素的偏移为0
- 负偏移索引意味着从最后或右边反向进行计数
- S[0]获取了第一个元素
- S[-2]获取了倒数第二元素（就像S[len(S)]-2一样）

```
In [79]: S = 'sliceofspam'
         S[0], S[-2]                # Indexing from front or end
```



字符串—索引和分片

- 分片

- 分片 ($S[i:j]$) 提取对应的部分作为一个序列:
- 上边界不包含在内, $S[i,j)$
- 如果没有给出的话, 分片的边界默认为0和序列的长度
 - $S[1:3]$ 获取了从偏移为1的元素, 直到但不包含偏移为3的元素
 - $S[1:]$ 获取了从偏移为1直到末尾 (偏移为序列长度) 之间的元素
 - $S[:3]$ 获取了从偏移为0直到但是不包括偏移为3之间的元素
 - $S[:-1]$ 获取了从偏移为0直到但是不包括最后一个元素之间的元素
 - $S[:]$ 获取了从偏移0到末尾之间的元素, 这有效实现了顶层S拷贝

```
print(S)
S[1:3], S[1:], S[:-1]      # Slicing: extract a section

sliceofspam
```

扩展分片：步进Stride

- 分片表达式还有个可选的第三个参数：步进(stride)。
- 完整形式的分片现在变成了 `S[l:j:k]`,
 - “索引对象 `S` 中的元素，从偏移为 `l` 直到偏移为 `j-1`，每隔 `k` 元素索引一次”。此处的 `k` 即为步进，默认值为1，这也是之前我们的切片中从左至右提取每一个元素的原因。
 - 如果定义了一个明确的步长值，那么能够使用第三个限制去跳过某些元素或反向排列它们的顺序
 - 步进值可以为负值

```
In [81]: S = 'abcdefghijklmnop'
         S[1:10:2]
```

```
Out[81]: 'bdfhj'
```

```
In [82]: S[::2]
```

```
Out[82]: 'acegikmo'
```

```
In [83]: S = 'hello'
         S[::-1]
```

```
Out[83]: 'olleh'
```

```
In [84]: S = 'abcdefg'
         S[5:1:-1]
```

```
Out[84]: 'fedc'
```


扩展分片：步进Stride

- 分片表达式还有个可选的第三个参数：步进(stride)。
 - 完整形式的分片现在变成了 `S[l:j:k]`,
 - “索引对象 `S` 中的元素，从偏移为 `l` 直到偏移为 `j-1`，每隔 `k` 元素索引一次”。此处的 `k` 即为步进，默认值为1，这也是之前我们的切片中从左至右提取每一个元素的原因。
 - 如果定义了一个明确的步长值，那么能够使用第三个限制去跳过某些元素或反向排列它们的顺序
 - 步进值可以为负值
- 在单次切片操作内，不要同时指定上边界、下边界和步进。 如果一定要用步进，那尽量采用正值*

```
In [81]: S = 'abcdefghijklmnop'
         S[1:10:2]
```

```
Out[81]: 'bdfhj'
```

```
In [82]: S[::2]
```

```
Out[82]: 'acegikmo'
```

```
In [83]: S = 'hello'
         S[::-1]
```

```
Out[83]: 'olleh'
```

```
In [84]: S = 'abcdefg'
         S[5:1:-1]
```

```
Out[84]: 'fedc'
```

字符串转换工具

- **int()** 函数可以将字符串转换为整数数字,
- **float()** 函数可以将字符串转换成浮点数,
- **str()** 函数可以将数字转换为字符串表达形式。

```
In [85]: S = '42'
         I = 1
         # S + I      # TypeError: cannot concatenate 'str' and 'int' objects
         int(S) + I
```

```
Out[85]: 43
```

```
In [86]: S + str(I)
```

```
Out[86]: '421'
```

```
In [87]: str(3.1415), float('1.5')
```

```
Out[87]: ('3.1415', 1.5)
```

```
In [88]: text = '1.234E-10'
         float(text)
```

```
Out[88]: 1.234e-10
```

字符串转换工具

- 单个的字符可以通过将其传给内置的 `ord()` 函数转换为其对应的 ASCII 码，而 `chr()` 函数将会执行相反的操作，将 ASCII 码转化为对应的字符：

```
In [89]: print(ord('s'))           # Return an integer of the given single Unicode character
          print(chr(115))          # Return a string of the given number

115
s
```

- 利用 `ord()` 和 `chr()` 函数，可以执行基于字符串的数学运算。

```
In [90]: S = '5'
          S = chr(ord(S) + 1)
          S
```

```
Out[90]: '6'
```

```
In [91]: S = chr(ord(S) + 1)
          S
```

```
Out[91]: '7'
```

字符串转换工具

- 将一个表示二进制的字符串转换为等值的十进制整数

- `B = '1101'`

```
B='1101'  
int(B, 2)           # Convert binary to integer: build-in function
```

13

```
B = '1101'  
I = 0  
while B != '':  
    I = I * 2 + (ord(B[0]) - ord('0'))  
    B = B[1:]  
print(I)
```

13

修改字符串

- Python 中字符串属于不可变序列。

```
S = 'spam'
S[0] = 'x'          # TypeError: 'str' object does not support item assignment

-----
TypeError                                Traceback (most recent call last)
<ipython-input-67-f280fd2c22d5> in <module>
      1 S = 'spam'
----> 2 S[0] = 'x'          # TypeError: 'str' object does not support item assignment

TypeError: 'str' object does not support item assignment
```

修改字符串

- Python 中字符串属于不可变序列。
- 若要改变一个字符串，需要利用合并、分片这样的工具来建立并赋值给一个新的字符串，倘若必要的话，还要将这个结果赋值给字符串最初的变量名

```
S = S + 'SPAM!'
S
'spamSPAM!'
```

```
S = S[:4] + 'Burger' + S[-1]
S
'spamBurger!'
```

```
S = 'splot'
S = S.replace('pl', 'pamal')
S
'spamalot'
```

字符串操作

• 查找和替换

- find() 和 replace()
- find() 方法搜索子字符串。
 - 该方法范围子字符串出现处的偏移或者未找到时返回 -1。

```
S = 'xxxxSPAMxxxxSPAMxxxx'
where = S.find('SPAM')           # Search for position
where                                # Occurs at offset 4
```

4

```
S = 'xxxxSPAMxxxxSPAMxxxx'
S.replace('SPAM', 'EGGS')        # Replace all

'xxxxEGGSxxxxEGGSxxxx'
```

```
S.replace('SPAM', 'EGGS', 1)     # Replace one

'xxxxEGGSxxxxSPAMxxxx'
```

字符串操作

- 文本解析

- `split()`

- 将一个字符串分割为一个子字符串的列表，以分隔符字符串为标准。如果没有传递分隔符，默认的分隔符为空格。

```
line = 'aaa bbb ccc'
cols = line.split()
cols
```

```
['aaa', 'bbb', 'ccc']
```

```
line = 'bob,hacker,40'      # 读取自csv文件
line.split(',')            # 使用逗号作为分隔符
```

```
['bob', 'hacker', '40']
```

```
line = "i'mSPAMaSPAMlumberjack"  #分隔符是一个字符串
line.split("SPAM")
```

```
["i'm", 'a', 'lumberjack']
```


字符串操作

- 大小写转换

- upper()
- lower()

```
line = "The knights who say Ni!\n"  
line.upper()          # Upper
```

```
'THE KNIGHTS WHO SAY NI!\n'
```

```
line.lower()          # Lower
```

```
'the knights who say ni!\n'
```

字符串操作-小结

| 方法 | 例子及含义 |
|-------------|---|
| S.find() | S.find('SPAM') 搜索子串，返回偏移 |
| S.replace() | S.replace('SPAM', 'EGGS', 1),用'EGGS'子串替换第一个'SPAM'子串 |
| S.split() | S.split(','), 用分隔符将一个字符串分割为一个子字符串的列表 |
| S.upper() | 转换大写 |
| S.lower() | 转换小写 |

字符串格式化 --- %表达式

- 字符串格式化有两种实现形式
- 字符串格式化表达式 %
 - 这是从 Python 诞生之初就有的方法

```
'That is %d %s bird!' % (1, 'dead')    # Format expression
```

```
'That is 1 dead bird!'
```

```
exclamation = 'Ni'
```

```
'The knights who say %s!' % exclamation # String substitution
```

```
'The knights who say Ni!'
```

```
'%d %s %g you' % (1, 'spam', 4.0)    # Type-specific substitutions
```

```
'1 spam 4 you'
```

```
'%s -- %s -- %s' % (42, 3.14159, [1, 2, 3]) # All types match a %s target
```

```
'42 -- 3.14159 -- [1, 2, 3]'
```

字符串格式化--- %表达式

- 对于特定类型的格式化，可以在格式化表达式中使用下表列出的转换代码

- 常用的%s, %d, %f。
- 格式化
- %e, %i, %g

| 代码 | 意义 |
|----|-------------------------|
| s | 字符串 (或者对象x转成str(x)的字符串) |
| r | 和s类似，但是使用repr, 而不是 str |
| c | 字符 (int or str) |
| d | 十进制整数 |
| i | 整数 |
| u | 与d一样 |
| o | 八进制整数 |
| x | 十六进制整数 |
| X | 与 x 一样，但是使用大写字母 |
| e | 浮点指数 |
| E | 与 e 一样，但是使用大写字母 |
| f | 浮点数 |
| F | 与 f 一样，但是使用大写字母 |
| g | 浮点数 e 或 f |
| G | 浮点数 E or F |
| % | 常量 % (等同于 %%) |

字符串格式化

```
x = 1234
res = 'integer: ...%d...%-6d...%06d' % (x, x, x)    # -:左对齐, 0:补零
res
```

```
'integer: ...1234...1234  ...001234'
```

```
y = 1.23456789
'%e | %f | %g' % (y, y, y)
```

```
'1.234568e+00 | 1.234568 | 1.23457'
```

```
x = 3.2223446e2
print('%5g' % (x))    #5=p, -4<=exp(-2)<p(5), use f, precision = 2 (p-1-exp)
print('%g' % (x))    #-4<=exp <p(5), use f, default width =6
```

```
x = 3.2223446e-5
print('%3g' % (x))    #3=p, use e, precision=3(p-1) format
print('%g' % (x))    #exp <-4, use e, default width =6
```

```
322.23
322.234
3.22e-05
3.22234e-05
```

字符串格式化

```
x = 1234
res = 'integer: ...%d...%-6d...%06d' % (x, x, x)    # -:左对齐, 0:补零
res
```

```
'integer: ...1234...1234  ...001234'
```

```
y = 1.23456789
'%e | %f | %g' % (y, y, y)
```

```
'1.234568e+00 | 1.234568 | 1.23457'
```

```
'%-6.2f | %05.2f | %+07.1f' % (y, y, y)    #字符串总长度.小数点后位数
```

```
'1.23    | 01.23 | +0001.2'
```

字符串格式化

```
x = 1234
res = 'integer: ...%d...%-6d...%06d' % (x, x, x)    # -:左对齐, 0:补零
res
```

```
'integer: ...1234...1234  ...001234'
```

```
y = 1.23456789
'%e | %f | %g' % (y, y, y)
```

```
'1.234568e+00 | 1.234568 | 1.23457'
```

```
'%-6.2f | %05.2f | %+07.1f' % (y, y, y)    #字符串总长度.小数点后位数
```

```
'1.23    | 01.23 | +0001.2'
```

- 通过引用字典中的键来提取相应的值

```
'%(qty)d more %(food)s' % {'qty':1, 'food':'spam'}
```

```
'1 more spam'
```

字符串格式化--- format()方法

- 字符串格式化第二种方法

- format() 方法
- 这是 Python 2.6 和 Python 3.x 新增加的方法。
- 和格式化%表达式不同，format() 方法的意图更加详细而明确。

```
: template = '{0}, {1} and {2}'           # By position
: template.format('spam', 'ham', 'eggs')
: 'spam, ham and eggs'
```

```
: template = '{motto}, {pork} and {food}' # By keyword
: template.format(motto='spam', pork='ham', food='eggs')
: 'spam, ham and eggs'
```

```
template = '{}', {} and {}'           # By relative position
template.format('spam', 'ham', 'eggs')
'spam, ham and eggs'
```


字符串格式化--- format()方法

- 调用format() 方法的也可以是一个临时字符串的常量，并且任意的对象类型都可以替换

```
template = '{motto}, {pork} and {food}'      # By keyword
print(template.format(motto='spam', pork='ham', food='eggs'))|
'{motto}, {pork} and {food}'.format(pork=42, motto=3.14, food=[1, 2])
```

spam, ham and eggs

实际代码中不建议交换位置次序

字符串格式化--- format()方法

- 调用format() 方法的也可以是一个临时字符串的常量，并且任意的对象类型都可以替换

```
template = '{motto}, {pork} and {food}'      # By keyword
print(template.format(motto='spam', pork='ham', food='eggs'))|
'{motto}, {pork} and {food}'.format(pork=42, motto=3.14, food=[1, 2])
```

spam, ham and eggs

'3.14, 42 and [1, 2]'

实际代码中不建议交换位置次序

- 字符串格式化方法中的方括号可以指定列表（或其他的序列）偏移量以执行索引，但是，只有单个正偏移值才有效。

```
somelist = list('SPAM')                      # Change string to list
'first={0[0]}, last={0[3]}'.format(somelist)
```

'first=S, last=M'

字符串格式化--- format()方法

- 字符格式控制

```
'{0:10} = {1:<10}'.format('spam', 123.4567)
```

```
'spam          = 123.4567 '
```

{0:10}意味着第一个位置将被格式化为10字符宽度

{1:<10}意味着第二个位置将被格式化为10字符宽度并且左对齐。

- 浮点数采用与 % 表达式中相同的代码类型和格式化声明。

```
'{0:.4f}'.format(1 / 3.0)
```

Parameters hardcoded, 位置0可以被省略

```
'0.3333'
```

```
# Floating-point numbers
```

```
'{0:e}, {1:.3e}, {2:f}'.format(3.14159, 3.14159, 3.14159)
```

```
'3.141590e+00, 3.142e+00, 3.141590'
```

字符串格式化--- format()方法

- 字符串格式化方法可以指定对象属性和字典键：

```
import sys
'My {0[kind]} runs {1.platform}'.format({'kind': 'laptop'}, sys)

'My laptop runs darwin'
```

```
eg_dict={'kind': 'laptop'}
'My {0[kind]} runs {1}'.format(eg_dict, sys.platform)

'My laptop runs darwin'
```

字符串格式化

- 尽量使用**format()**格式化方法
 - 有一个更容易记忆的方法名
 - 拥有表达式 % 所没有的一些额外功能
 - 可以更明确地进行替代值引用
 - 表达式 % 在将来可能被废弃

字符串小结

- Python 中的字符串不能改变
- Python 中单引号和双引号使用完全相同
- 使用三引号(''或''')可以指定一个多行字符串
- 转义符 \
- 反斜杠可以用来转义，使用r可以让反斜杠不发生转义。
 - 如 r"this is a line with \n" 则\n会显示，并不是换行
- 按字面意义级联字符串，如"this " "is " "string"会被自动转换为this is string
- 字符串可以用 + 运算符连接在一起，用 * 运算符重复

字符串小结

- Python 中的字符串有两种索引方式，从左往右以 0 开始，从右往左以 -1 开始
- Python 没有单独的 char 字符类型，一个字符就是长度为 1 的字符串
- 字符串的截取的语法格式如下：变量[头下标:尾下标:步长]
- 可以用表达式 % 或方法 format() 进行字符串格式化

练习

- 字符串'a\nb\x1f\000d'共有多少个字符？
- 已知字符串S的值为's,pa,m'，如何从中间抽取两个字符？
- 如何将整数字符串转为二进制字符串？