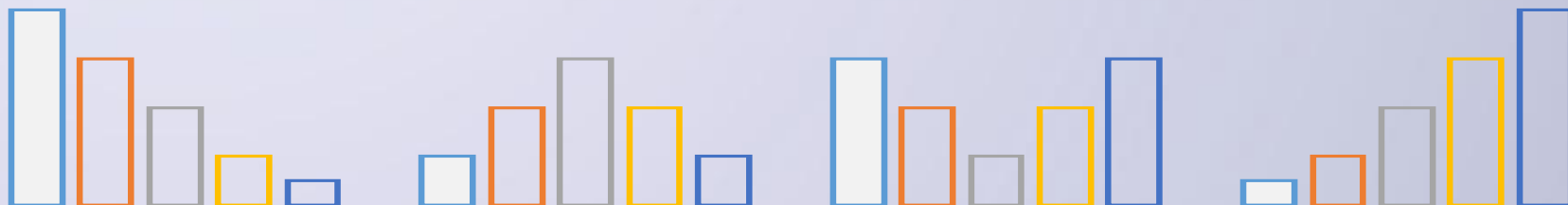
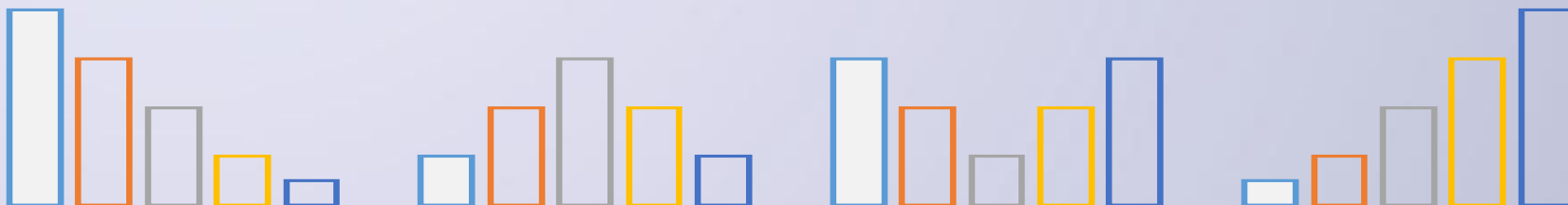


# Python语言程序设计

北京理工大学 嵩天



# 第3章 基本数据类型





# 数字类型



# 数字类型

- 程序元素：010/10，存在多种可能
  - 表示十进制整数值10
  - 类似人名一样的字符串
- 数字类型对Python语言中数字的表示和使用  
进行了定义和规范



# 数字类型

Python语言包括三种数字类型

- 整数类型
- 浮点数类型
- 复数类型



# 整数类型

- 与数学中的整数概念一致，没有取值范围限制
- `pow(x, y)`函数：计算 $x^y$
- 打开IDLE
  - 程序1：`pow(2,10)` , `pow(2,15)`
  - 程序2：`pow(2, 1000)`
  - 程序3：`pow(2, pow(2,15))`



# 整数类型


## ■ 示例

■ 1010, 99, -217

■ 0x9a, -0X89 (0x, 0X开头表示16进制数)

■ 0b010, -0B101 (0b, 0B开头表示2进制数)

■ 0o123, -0O456 (0o, 0O开头表示8进制数)



# 浮点数类型

- 带有小数点及小数的数字
- Python语言中浮点数的数值范围存在限制，小数精度也存在限制。这种限制与在不同计算机系统有关





# 浮点数类型

```
>>> import sys
>>> sys.float_info
sys.float_info(max=1.7976931348623157e+308, max_exp
=1024, max_10_exp=308, min=2.2250738585072014e-308,
min_exp=-1021, min_10_exp=-307, dig=15, mant_dig=53
, epsilon=2.220446049250313e-16, radix=2, rounds=1)
>>>
```



# 浮点数类型

## ■ 示例

■ 0.0, -77., -2.17

■ 96e4, 4.3e-3, 9.6E5 （科学计数法）

■ 科学计数法使用字母 “e” 或者 “E” 作为幂的符号，以10为基数。科学计数法含义如下：

$$<a>e<b> = a * 10^b$$



# 复数类型

- 与数学中的复数概念一致,  $z = a + bj$ ,  $a$ 是实数部分,  $b$ 是虚数部分,  $a$ 和 $b$ 都是浮点类型, 虚数部分用 $j$ 或者 $J$ 标识

- 示例：

$12.3 + 4j$ ,  $-5.6 + 7j$




# 复数类型

- $z = 1.23e-4 + 5.6e+89j$  ( 实部和虚部是什么 ? )
- 对于复数  $z$  , 可以用 `z.real` 获得实数部分 ,  
`z.imag` 获得虚数部分
- `z.real = 0.000123`    `z.imag = 5.6e+89`



# 数字类型的操作



# 内置的数值运算操作符

- 三种类型存在一种逐渐“扩展”的关系：

整数 -> 浮点数 -> 复数

( 整数是浮点数特例，浮点数是复数特例 )

- 不同数字类型之间可以进行混合运算，运算后生成结果为最宽类型

- $123 + 4.0 = 127.0$  (整数 + 浮点数 = 浮点数)



# 内置的数值运算操作符

| 操作符      | 描述                      |
|----------|-------------------------|
| $x + y$  | x与y之和                   |
| $x - y$  | x与y之差                   |
| $x * y$  | x与y之积                   |
| $x / y$  | x与y之商                   |
| $x // y$ | x与y之整数商，即：不大于x与y之商的最大整数 |
| $x \% y$ | x与y之商的余数，也称为模运算         |
| $-x$     | x的负值，即： $x*(-1)$        |
| $+x$     | x本身                     |
| $x**y$   | x的y次幂，即： $x^y$          |



# 内置的数值运算操作符

数字类型之间相互运算所生成的结果是“更宽”的类型，基本规则是：

- 整数之间运算，如果数学意义上的结果是小数，结果是浮点数；
- 整数之间运算，如果数学意义上的结果是整数，结果是整数；
- 整数和浮点数混合运算，输出结果是浮点数；
- 整数或浮点数与复数运算，输出结果是复数。





# 内置的数值运算函数

Python解释器提供了一些内置函数，在这些内置函数之中，有6个函数与数值运算相关

| 函数   | 描述  |
|--|---|
| <code>abs(x)</code>  | x的绝对值   |
| <code>divmod(x, y)</code>  | $(x//y, x\%y)$ ，输出为二元组形式（也称为元组类型）   |
| <code>pow(x, y[, z])</code>  | $(x**y)\%z$ ， <code>[..]</code> 表示该参数可以省略，即： <code>pow(x,y)</code> ，它与 <code>x**y</code> 相同 |
| <code>round(x[, ndigits])</code>                                   | 对x四舍五入，保留ndigits位小数。 <code>round(x)</code> 返回四舍五入的整数值                                       |
| <code>max(x<sub>1</sub>, x<sub>2</sub>, ..., x<sub>n</sub>)</code> | x <sub>1</sub> , x <sub>2</sub> , ..., x <sub>n</sub> 的最大值，n没有限定                            |
| <code>min(x<sub>1</sub>, x<sub>2</sub>, ..., x<sub>n</sub>)</code> | x <sub>1</sub> , x <sub>2</sub> , ..., x <sub>n</sub> 的最小值，n没有限定                            |



# 数字类型的转换

数值运算操作符可以隐式地转换输出结果的数字类型

例如，两个整数采用运算符 “/” 的除法将可能输出浮点数结果。

此外，通过内置的数字类型转换函数可以显式地在数字类型之间进行转换

| 函数                             | 描述  |
|--------------------------------|---|
| <code>int(x)</code>            | 将x转换为整数，x可以是浮点数或字符串                                   |
| <code>float(x)</code>          | 将x转换为浮点数，x可以是整数或字符串                                   |
| <code>complex(re[, im])</code> | 生成一个复数，实部为re，虚部为im，re可以是整数、浮点数或字符串，im可以是整数或浮点数但不能为字符串 |



# 数字类型的转换

- 三种类型可以相互转换


函数：`int()`, `float()`, `complex()`

- 示例：

- `int(4.5) = 4` （直接去掉小数部分）

- `float(4) = 4.0` （增加小数部分）


- `complex(4) = 4 + 0j`



# 数字类型的转换

- 示例： $\text{complex}(4.5) = 4.5 + 0j$

```
>>> float(4.5+0j)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    float(4.5+0j)
TypeError: can't convert complex to float
>>>
```

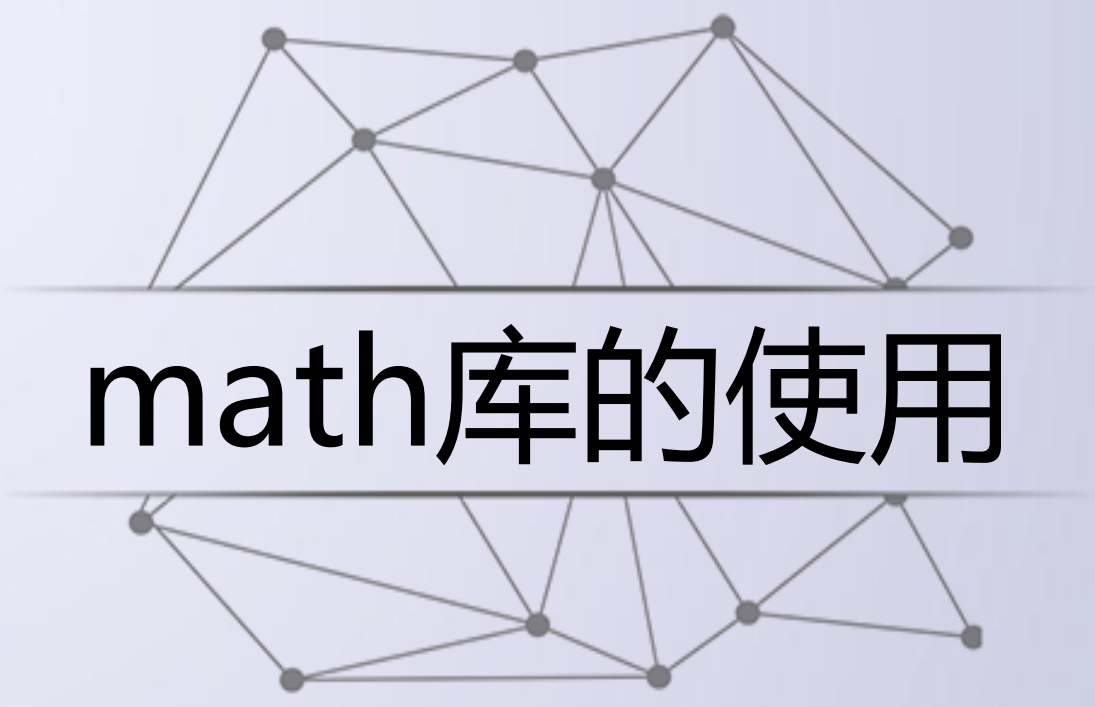


# 数字类型的判断

- 函数：type(x)，返回x的类型，适用于所有类型的判断

- 示例：

```
>>> type(4.5)
<class 'float'>
>>> type(z)
<class 'complex'>
>>>
```



# math库的使用



# math库概述

- math库是Python提供的内置数学类函数库
- math库不支持复数类型
- math库一共提供了4个数学常数和44个函数。
  - 44个函数共分为4类，包括：16个数值表示函数、8个幂对数函数、16个三角对数函数和4个高等特殊函数



# math库概述

首先使用保留字import引用该库

- 第一种：import math

对math库中函数采用math.<b>()形式使用

```
>>>import math
>>>math.ceil(10.2)
11
```

- 第二种，from math import <函数名>

对math库中函数可以直接采用<函数名>()形式使用

```
>>>from math import floor
>>>floor(10.2)
10
```





# math库解析

## ■ math库包括4个数学常数

| 常数       | 数学表示     | 描述                       |
|----------|----------|--------------------------|
| math.pi  | $\pi$    | 圆周率，值为3.141592653589793  |
| math.e   | $e$      | 自然对数，值为2.718281828459045 |
| math.inf | $\infty$ | 正无穷大，负无穷大为-math.inf      |
| math.nan |          | 非浮点数标记，NaN（Not a Number） |



# math库解析

## ■ math库包括16个数值表示函数

| 函数                   | 数学表示                | 描述                                    |
|----------------------|---------------------|---------------------------------------|
| math.fabs(x)         | $ x $               | 返回x的绝对值                               |
| math.fmod(x, y)      | $x \% y$            | 返回x与y的模                               |
| math.fsum([x,y,...]) | $x+y+\dots$         | 浮点数精确求和                               |
| math.ceil(x)         | $\lceil x \rceil$   | 向上取整, 返回不小于x的最小整数                     |
| math.floor(x)        | $\lfloor x \rfloor$ | 向下取证, 返回不大于x的最大整数                     |
| math.factorial(x)    | $x!$                | 返回x的阶乘, 如果x是小数或负数, 返回ValueError       |
| math.gcd(a, b)       |                     | 返回a与b的最大公约数                           |
| math.frexp(x)        | $x = m * 2^e$       | 返回(m, e), 当x=0, 返回(0.0, 0)            |
| math.ldexp(x, i)     | $x * 2^i$           | 返回 $x * 2^i$ 运算值, math.frexp(x)函数的反运算 |
| math.modf(x)         |                     | 返回x的小数和整数部分                           |
| math.trunc(x)        |                     | 返回x的整数部分                              |
| math.copysign(x, y)  | $ x  *  y /y$       | 用数值y的正负号替换数值x的正负号                     |
| math.isclose(a,b)    |                     | 比较a和b的相似性, 返回True或False               |
| math.isfinite(x)     |                     | 当x为无穷大, 返回True; 否则, 返回False           |
| math.isinf(x)        |                     | 当x为正数或负数无穷大, 返回True; 否则, 返回False      |
| math.isnan(x)        |                     | 当x是NaN, 返回True; 否则, 返回False           |



# math库解析

## ■ math库中包括8个幂对数函数

| 函数                              | 数学表示            | 描述                             |
|---------------------------------|-----------------|--------------------------------|
| <code>math.pow(x,y)</code>      | $x^y$           | 返回x的y次幂                        |
| <code>math.exp(x)</code>        | $e^x$           | 返回e的x次幂，e是自然对数                 |
| <code>math.expml(x)</code>      | $e^x-1$         | 返回e的x次幂减1                      |
| <code>math.sqrt(x)</code>       | $\sqrt{x}$      | 返回x的平方根                        |
| <code>math.log(x[,base])</code> | $\log_{base} x$ | 返回x的对数值，只输入x时，返回自然对数，即 $\ln x$ |
| <code>math.log1p(x)</code>      | $\ln(1+x)$      | 返回1+x的自然对数值                    |
| <code>math.log2(x)</code>       | $\log x$        | 返回x的2对数值                       |
| <code>math.log10(x)</code>      | $\log_{10} x$   | 返回x的10对数值                      |



# math库解析

## ■ math库包括六个 “三角双曲函数”

| 函数              | 数学表示                       | 描述                   |
|-----------------|----------------------------|----------------------|
| math.degree(x)  |                            | 角度x的弧度值转角度值          |
| math.radians(x) |                            | 角度x的角度值转弧度值          |
| math.hypot(x,y) | $\sqrt{x^2 + y^2}$         | 返回(x,y)坐标到原点(0,0)的距离 |
| math.sin(x)     | $\sin x$                   | 返回x的正弦函数值，x是弧度值      |
| math.cos(x)     | $\cos x$                   | 返回x的余弦函数值，x是弧度值      |
| math.tan(x)     | $\tan x$                   | 返回x的正切函数值，x是弧度值      |
| math.asin(x)    | $\arcsin x$                | 返回x的反正弦函数值，x是弧度值     |
| math.acos(x)    | $\arccos x$                | 返回x的反余弦函数值，x是弧度值     |
| math.atan(x)    | $\arctan x$                | 返回x的反正切函数值，x是弧度值     |
| math.atan2(y,x) | $\arctan y/x$              | 返回y/x的反正切函数值，x是弧度值   |
| math.sinh(x)    | $\sinh x$                  | 返回x的双曲正弦函数值          |
| math.cosh(x)    | $\cosh x$                  | 返回x的双曲余弦函数值          |
| math.tanh(x)    | $\tanh x$                  | 返回x的双曲正切函数值          |
| math.asinh(x)   | $\operatorname{arcsinh} x$ | 返回x的反双曲正弦函数值         |
| math.acosh(x)   | $\operatorname{arccosh} x$ | 返回x的反双曲余弦函数值         |
| math.atanh(x)   | $\operatorname{arctanh} x$ | 返回x的反双曲正切函数值         |



# math库解析

## ■ math库包括4个高等特殊函数

| 函数                          | 数学表示   | 描述  |
|-----------------------------|--|---|
| <code>math.erf(x)</code>    | $\frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$        | 高斯误差函数，应用于概率论、统计学等领域                                |
| <code>math.erfc(x)</code>   | $\frac{2}{\sqrt{\pi}} \int_x^{\infty} e^{-t^2} dt$ | 余补高斯误差函数， <code>math.erfc(x)=1 - math.erf(x)</code> |
| <code>math.gamma(x)</code>  | $\int_0^{\infty} x^{t-1} e^{-x} dx$                | 伽玛（Gamma）函数，也叫欧拉第二积分函数                              |
| <code>math.lgamma(x)</code> | <code>ln(gamma(x))</code>                          | 伽玛函数的自然对数   |



# 实例3: 天天向上的力量



## 实例代码3.1: 天天向上

一年365天，以第1天的能力值为基数，记为1.0，当好好学习时能力值相比前一天提高1‰，当没有学习时由于遗忘等原因能力值相比前一天下降1‰。每天努力和每天放任，一年下来的能力值相差多少呢？



# 实例代码3.1: 天天向上

实例代 码3.1  
e3.1DayDayUp365.py

```
1 #e3.1DayDayUp365.py
2 import math
3 dayup = math.pow((1.0 + 0.001), 365) # 提高0.001
4 daydown = math.pow((1.0 - 0.001), 365) # 放任0.001
5 print("向上: {:.2f}, 向下: {:.2f}.".format(dayup, daydown))
```


■ 运行结果如下，每天努力1‰，一年下来将提高44%，好像不多？请继续分析。





## 实例代码3.2: 天天向上

一年365天，如果好好学习时能力值相比前一天提高5‰，当放任时相比前一天下降5‰。  
效果相差多少呢？



## 实例代码3.2: 天天向上

实例代 e3.2DayDayUp365.py  
码3.2

```
1 #e3.2DayDayUp365.py
2 import math
3 dayup = math.pow((1.0 + 0.005), 365) # 提高0.005
4 daydown = math.pow((1.0 - 0.005), 365) # 放任0.005
5 print(" 向 上 : {:.2f}, 向 下 : {:.2f}.".format(dayup,
daydown))
```

■ 运行结果如下，每天努力5‰，一年下来将提高6倍！这不容小觑了吧？



## 实例代码3.3: 天天向上

一年365天，如果好好学习时能力值相比前一天提高1%，当放任时相比前一天下降1%。效果相差多少呢？



# 实例代码3.3: 天天向上

实例代 码3.3 e3.3DayDayUp365.py


```
#e3.3DayDayUp365.py
import math
1 dayfactor = 0.01
2 dayup = math.pow((1.0 + dayfactor), 365) # 提高
3 dayfactor
4 daydown = math.pow((1.0 - dayfactor), 365) # 放任
5 dayfactor
6 print(" 向上 : {:.2f}, 向下 : {:.2f}.".format(dayup,
daydown))
```

- 运行结果如下，每天努力1%，一年下来将提高37倍。这个相当惊人吧！



## 实例代码3.4: 天天向上

一年365天，一周5个工作日，如果每个工作日都很努力，可以提高1%，仅在周末放任一下，能力值每天下降1%，效果如何呢？



# 实例代码3.4: 天天向上

实例代 码3.4  
e3.4DayDayUp365.py

```
1  #e3.4DayDayUp365.py
2  dayup, dayfactor = 1.0, 0.01
3  for i in range(365):
4      if i % 7 in [6, 0]:#周六周日
5          dayup = dayup * (1 - dayfactor)
6      else:
7          dayup = dayup * (1 + dayfactor)
8      print("向上5天向下2天的力量: {:.2f}".format(dayup))
```

■ 猜猜运行结果？每周努力5天，而不是每天，一年下来，水平仅是初始的4.63倍！与每天坚持所提高的237倍相去甚远



## 实例代码3.5: 天天向上

如果对实例代码3.4的结果感到意外，那自然会产生如下问题：每周工作5天，休息2天，休息日水平下降0.01，工作日要努力到什么程度一年后的水平才与每天努力1%所取得的效果一样呢？

# 实例代码3.5: 天天向上

■ 如果每周连续努力5天，休息2天，为了达到每天努力1%所达到的水平，则需要在工作日将提高的程度达到约2%，即要努力1倍才仅是为了休息2天。

■ 这就是天天向上的力量！

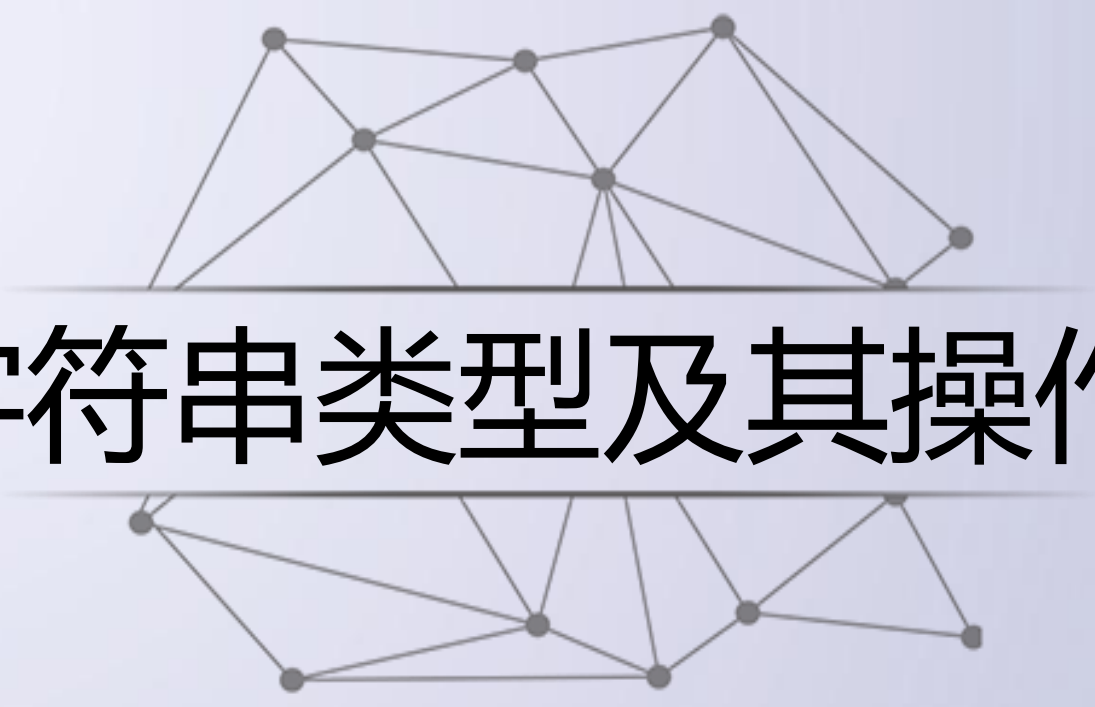
实例 e3.5DayDayUp365.py

代码


3.5

```
1 #e3.5DayDayUp365.py
2 def dayUP(df):
3     dayup = 0.01
4     for i in range(365):
5         if i % 7 in [6, 0]:
6             dayup = dayup * (1 - 0.01 * df)
7         else:
8             dayup = dayup * (1 + df)
9     return dayup
10 dayfactor = 0.01
11 while (dayUP(dayfactor) < 37.78):
12     dayfactor += 0.001
13 print("每天的努力参数是: {:.3f}.".format(dayfactor))
```





# 字符串类型及其操作



# 字符串类型


- 字符串是用双引号"或者单引号"括起来的一个或多个字符。
- 字符串可以保存在变量中，也可以单独存在。
- 可以用type()函数测试一个字符串的类型



# 字符串类型

- Python语言转义符：\
- 输出带有引号的字符串，可以使用转义符
- 使用 \\ 输出带有转移符的字符串

```
>>> print ("\"大家好\"")  
"大家好"  
>>>
```



# 字符串类型

- 字符串是一个字符序列：字符串最左端位置标记为0，依次增加。字符串中的编号叫做“索引”

|   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|
| H | e | l | l | o |   | J | o | h | n |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |



# 字符串类型

- 单个索引辅助访问字符串中的特定位置

格式为<string>[<索引>]

```
>>> greet="Hello John"
>>> print(greet[2])
l
>>> x=8
>>> print(greet[x-2])
J
>>>
```



# 字符串类型

- Python中字符串索引从0开始，一个长度为L的字符串最后一个字符的位置是L-1
- Python同时允许使用负数从字符串右边末尾向左边进行反向索引，最右侧索引值是-1

```
>>> greet[-4]  
'J'  
>>>
```




# 字符串类型

- 可以通过两个索引值确定一个位置范围，返回这个范围的子串

格式： <string>[<start>:<end>]

- start和end都是整数型数值，这个子序列从索引start开始直到索引end结束，但不包括end位置。

```
>>> greet[0:3]
'Hel'
>>>
```



# 字符串类型

- 字符串之间可以通过+或\*进行连接
  - 加法操作(+)将两个字符串连接成为一个新的字符串
  - 乘法操作(\*)生成一个由其本身字符串重复连接而成的字符串

```
>>> "pine" + "apple"
'pineapple'
>>> 3 * "pine"
'pinepinepine'
>>>
```





# 字符串类型

- len()函数能否返回一个字符串的长度

```
>>> len("pine")  
4  
>>> len("祖国，您好！")  
6
```



# 字符串类型的转换

- 大多数数据类型都可以通过str()函数转换为字符串

```
>>> str(123)
'123'
>>> str(123.456)
'123.456'
>>> str(123e+10)
'1230000000000.0'
```



# 字符串使用实例

输入一个月份数字，返回对应月份名称缩写

这个问题的IPO模式是：

输入：输入一个表示月份的数字(1-12)

处理：利用字符串基本操作实现该功能

输出：输入数字对应月份名称的缩写



# 字符串使用实例

- 将所有月份名称缩写存储在字符串中

```
months = "JanFebMarAprMayJunJulAugSepOctNovDec"
```

- 在字符串中截取适当的子串来查找特定月份
  - 找出在哪里切割子串
  - 每个月份的缩写都由3个字母组成，如果pos表示一个月份的第一个字母，则months[pos:pos+3]表示这个月份的缩写，即：
    - `monthAbbrev = months[pos:pos+3]`



# 字符串使用实例

|     | 月份 | 字符串中位置 |
|-----|----|--------|
| Jan | 1  | 0      |
| Feb | 2  | 3      |
| Mar | 3  | 6      |
| Apr | 4  | 9      |



# 字符串使用实例

```
# month.py
months="JanFebMarAprMayJunJulAugSepOctNovDec"
n=input("请输入月份数(1-12):")
pos=(int(n)-1) * 3
monthAbbrev=months[pos:pos+3]
print("月份简写是"+monthAbbrev+".")
```

```
>>>
请输入月份数(1-12):7
月份简写是Jul.
>>>
```



# 字符串的操作

可以通过 for 和 in 组成的循环来遍历字符串中每个字符

■格式如下：

```
for <var> in <string>:
```

    操作



# 字符串的操作

用转义符可以在字符串中表达一些不可直接打印的信息

例如：用\n表示换行

- 字符串"Hello\nWorld\n\nGoodbye 32\n"

- 用print()函数打印后的输出效果如下：

Hello

World

Goodbye 32





# 内置的字符串处理函数

| 操作                    | 含义            |
|-----------------------|---------------|
| +                     | 连接            |
| *                     | 重复            |
| <string>[ ]           | 索引            |
| <string>[:]           | 剪切            |
| len(<string>)         | 长度            |
| <string>.upper()      | 字符串中字母大写      |
| <string>.lower()      | 字符串中字母小写      |
| <string>.strip()      | 去两边空格及去指定字符   |
| <string>.split()      | 按指定字符分割字符串为数组 |
| <string>.join()       | 连接两个字符串序列     |
| <string>.find()       | 搜索指定字符串       |
| <string>.replace()    | 字符串替换         |
| for <var> in <string> | 字符串迭代         |



# 内置的字符串处理函数

## 微实例3.2：恺撒密码。

凯撒密码是古罗马凯撒大帝用来对军事情报进行加密的算法，它采用了替换方法对信息中的每一个英文字符循环替换为字母表序列该字符后面第三个字符，对应关系如下：

原文：A B C D E F G H I J K L M N O P Q R S T U V W X Y Z


密文：D E F G H I J K L M N O P Q R S T U V W X Y Z A B C

原文字符P，其密文字符C满足如下条件：

$$C = (P + 3) \bmod 26$$

解密方法反之，满足：

$$P = (C - 3) \bmod 26$$



# 内置的字符串处理函数

微实例      m3.2 CaesarCode.py

3.2

```
1 plaincode = input("请输入明文: ")
2 for p in plaincode:
3     if ord("a") <= ord(p) <= ord("z"):
4         print(chr(ord("a")+(ord(p) -
5 ord("a")+3)%26), end='')
6     else:
7         print(p, end='')
```

微实例运行结果如下:

```
>>>
```

```
请输入明文: python is an excellent language.
```

```
sbwkrq lv dq hafhoohqw odqjxdjh.
```



# 内置的字符串处理方法

| 方法  | 描述  |
|---|---|
| <code>str.lower()</code>                            | 返回字符串 <code>str</code> 的副本，全部字符小写   |
| <code>str.upper()</code>                            | 返回字符串 <code>str</code> 的副本，全部字符大写   |
| <code>str.islower()</code>                          | 当 <code>str</code> 所有字符都是小写时，返回 <code>True</code> ，否则 <code>False</code>  |
| <code>str.isprintable()</code>                      | 当 <code>str</code> 所有字符都是可打印的，返回 <code>True</code> ，否则 <code>False</code>   |
| <code>str.isnumeric()</code>                        | 当 <code>str</code> 所有字符都是字符时，返回 <code>True</code> ，否则 <code>False</code>  |
| <code>str.isspace()</code>                          | 当 <code>str</code> 所有字符都是空格，返回 <code>True</code> ，否则 <code>False</code>   |
| <code>str.endswith(suffix[,start[,end]])</code>     | <code>str[start: end]</code> 以 <code>suffix</code> 结尾返回 <code>True</code> ，否则返回 <code>False</code>  |
| <code>str.startswith(prefix[, start[, end]])</code> | <code>str[start: end]</code> 以 <code>prefix</code> 开始返回 <code>True</code> ，否则返回 <code>False</code>  |
| <code>str.split(sep=None, maxsplit=-1)</code>       | 返回一个列表，由 <code>str</code> 根据 <code>sep</code> 被分割的部分构成  |
| <code>str.count(sub[,start[,end]])</code>           | 返回 <code>str[start: end]</code> 中 <code>sub</code> 子串出现的次数  |
| <code>str.replace(old, new[, count])</code>         | 返回字符串 <code>str</code> 的副本，所有 <code>old</code> 子串被替换为 <code>new</code> ，如果 <code>count</code> 给出，则前 <code>count</code> 次 <code>old</code> 出现被替换 |
| <code>str.center(width[, fillchar])</code>          | 字符串居中函数，详见函数定义  |
| <code>str.strip([chars])</code>                     | 返回字符串 <code>str</code> 的副本，在其左侧和右侧去掉 <code>chars</code> 中列出的字符  |
| <code>str.zfill(width)</code>                       | 返回字符串 <code>str</code> 的副本，长度为 <code>width</code> ，不足部分在左侧添0  |
| <code>str.format()</code>                           | 返回字符串 <code>str</code> 的一种排版格式，3.6节将详细介绍  |
| <code>str.join(iterable)</code>                     | 返回一个新字符串，由组合数据类型（见第6章） <code>iterable</code> 变量的每个元素组成，元素间用 <code>str</code> 分割   |



# 字符串类型的格式化



# format()方法的基本使用

字符串format()方法的基本使用格式是：

<模板字符串>.format(<逗号分隔的参数>)

```
"{ }： 计算机{ }的CPU占用率为{ }%。".format("2016-12-31", "PYTHON", 10)
```

↑  
0

↑  
1

↑  
2

字符串中槽{}的顺序

↑  
0

↑  
1

↑  
2

format() 中参数的顺序



# format()方法的基本使用

`"{1}: 计算机{0}的CPU占用率为{2}%。".format("2016-12-31", "PYTHON", 10)`

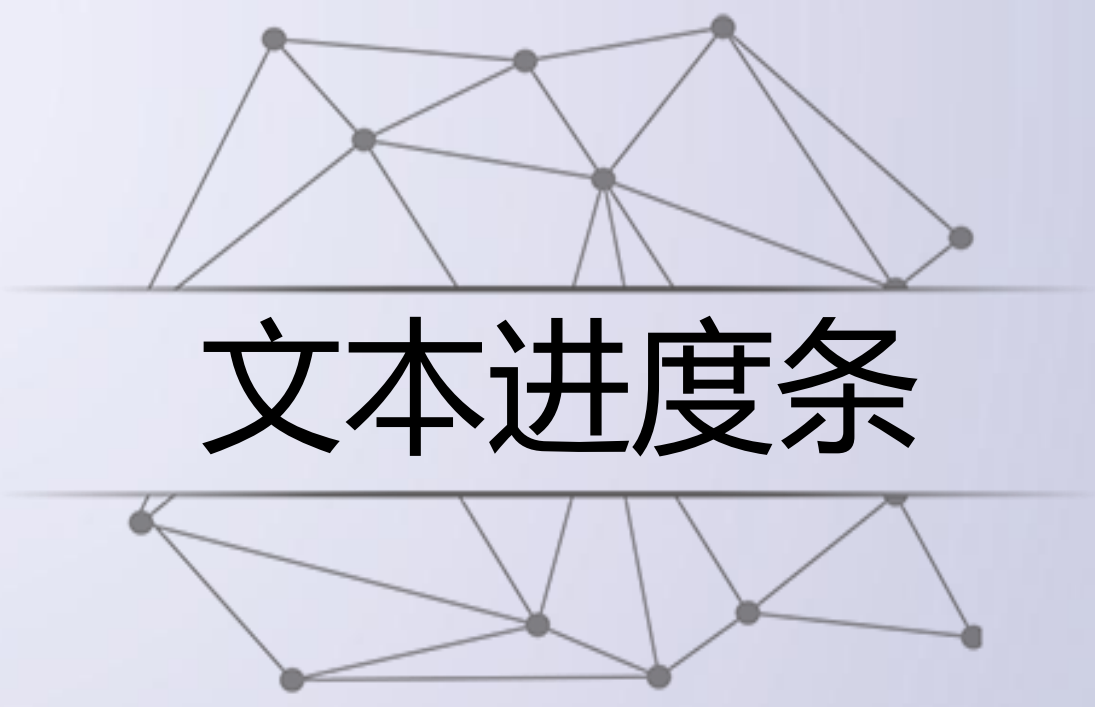


# format()方法的格式控制


- `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>}`
- 其中，格式控制标记用来控制参数显示时的格式。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，这里按照使用方式逐一介绍。

| :    | <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<br>> 右对齐<br>^ 居中对齐 | 槽的设定输出宽度 | 数字的千位分隔符<br>适用于整数和浮点数 | 浮点数小数部分的精度<br>或字符串的最大输出长度 | 整数类型<br>b, c, d, o, x, X<br>浮点数类型<br>e, E, f, % |



An abstract geometric design featuring two horizontal lines. Between the lines, there are two clusters of triangles and dots. The top cluster is a complex network of interconnected triangles and dots, with some dots acting as vertices. The bottom cluster is a simpler network of triangles and dots, also interconnected. The entire design is rendered in a light gray color on a white background.

# 文本进度条



# 简单的开始

利用print()函数实现简单的非刷新文本进度条

基本思想是按照任务执行百分比将整个任务划分为100个单位，每执行N%输出一次进度条。每一行输出包含进度百分比，代表已完成的部分(\*\*)和未完成的部分(..)的两种字符，以及一个跟随完成度前进的小箭头，风格如下：

```
%10 [*****->.....]
```



# 简单的开始

实例代 e4.1TextProgressBar.py  
码4.1

```
1  #e4.1TextProgressBar.py
2  import time
3  scale = 10
4  print("-----执行开始-----")
5  for i in range(scale+1):
6      a, b = '**' * i, '..' * (scale - i)
7      c = (i/scale)*100
8      print("%{:^3.0f}[{}->{}]" .format (c, a, b))
9      time.sleep(0.1)
10 print("-----执行结束-----")
```



# 简单的开始

程序的输出效果如下图：

```
-----执行开始-----  
% 0 [->.....]  
%10 [**->.....]  
%20 [****->.....]  
%30 [*****->.....]  
%40 [*****->.....]  
%50 [*****->.....]  
%60 [*****->.....]  
%70 [*****->.....]  
%80 [*****->....]  
%90 [*****->..]  
%100 [*****->]  
-----执行结束-----
```



# 单行动态刷新

实例代 码4.2 e4.2TextProgressBar.py

```
1 #e4.2TextProgressBar.py
2 import time
3 for i in range(101):
4     print("\r{:2}%".format(i), end="")
5     time.sleep(0.05)
```



# 单行动态刷新

上述程序在IDLE中的执行效果如下。

```
>>>
 0%  1%  2%  3%  4%  5%  6%  7%  8%  9% 10% 11% 12% 13% 14% 15% 16%
17% 18% 19% 20% 21% 22% 23% 24% 25% 26% 27% 28% 29% 30% 31% 32% 33%
34% 35% 36% 37% 38% 39% 40% 41% 42% 43% 44% 45% 46% 47% 48% 49% 50%
51% 52% 53% 54% 55% 56% 57% 58% 59% 60% 61% 62% 63% 64% 65% 66% 67%
68% 69% 70% 71% 72% 73% 74% 75% 76% 77% 78% 79% 80% 81% 82% 83% 84%
85% 86% 87% 88% 89% 90% 91% 92% 93% 94% 95% 96% 97% 98% 99%100%
```

- 为什么输出没有单行刷新呢？
- 这是因为IDLE本身屏蔽了单行刷新功能，如果希望获得刷新效果，请使用控制台命令行执行e4.2TextProgressBar.py程序。以Windows系统为例，启动命令行工具（<Windows系统安装目录>\system32\cmd.exe），选择到e4.2TextProgressBar.py文件所在目录执行



# 带刷新的文本进度条

实例代 e4.3TextProgressBar.py

码4.3

```
1  #e4.3TextProgressBar.py
2  import time
3  scale = 50
4  print("执行开始".center(scale//2, '-'))
5  t = time.clock()
6  for i in range(scale+1):
7      a = '*' * i
8      b = '.' * (scale - i)
9      c = (i/scale)*100
10     t -= time.clock()
11     print("\r{:^3.0f}%[{}->{}]{:.2f}s".format(c,a,b,-
12     t),\
13     end='')
14     time.sleep(0.05)
15 print("\n"+"执行结束".center(scale//2, '-'))
```



# 带刷新的文本进度条

采用命令行的执行上述文件，效果如下：


```
: \>python e4.3TextProgressBar.py
```

```
-----执行开始-----
```

```
100%[*****->] 65.71s
```

```
-----执行结束-----
```





# 五花八门的进度条设计函数

| 设计名称            | 趋势         | 设计函数  |
|-----------------|------------|---|
| Liner           | Constant   | $f(x) = x$  |
| Early Pause     | Speeds up  | $f(x) = x + (1 - \sin(x * \pi * 2 + \pi / 2)) / -8$ |
| Late Pause      | Slows down | $f(x) = x + (1 - \sin(x * \pi * 2 + \pi / 2)) / 8$  |
| Slow Wavy       | Constant   | $f(x) = x + \sin(x * \pi * 5) / 20$                 |
| Fast Wavy       | Constant   | $f(x) = x + \sin(x * \pi * 20) / 80$                |
| Power           | Speeds up  | $f(x) = (x + (1 - x) * 0.03)^2$                     |
| Inverse Power   | Slows down | $f(x) = 1 + (1 - x)^{1.5} * -1$                     |
| Fast Power      | Speeds up  | $f(x) = (x + (1 - x) / 2)^8$                        |
| Inv. Fast Power | Slows down | $f(x) = 1 + (1 - x)^3 * -1$                         |