1. 基本数据类型
2. 数字类型及操作

整数类型：与数学中的整数的概念一致，可正可负，没有取值范围限制

pow(x,y)函数：计算x的y次方，想算多大算多大

十进制、二进制：以0b或0B开头、 八进制：以0O或0o开头、 十六进制：以0x或0X开头。

浮点数类型：与数学中的实数概念一致，带有小数点及小数的数字。浮点数取值范围和小数精度都存在限制，但常规计算可忽略。 取值范围数量级约-10的308次方至10的308次方，精度数量级10的-16次方。

浮点数间运算存在不确定尾数，不是bug。

计算机中用53位二进制表示小数部分，约10的-16次方。二进制表示小数，可以无限接近，但不完全相同。

例如：0.1+0.2 结果无限接近0.3，但可能存在尾数

round（x，d）：对x四舍五入，d是小数截取位数。浮点数间运算及比较用round（）函数辅助。不确定尾数一般发生在10的-16次方左右，round（）十分有效。

浮点数可以采用科学计数法表示，使用字母e或E作为幂的符号，以10为基数，

格式：<a>e<b> 表示 a\*10的b次方。 例如：4.3e-3的值为0.0043

复数类型：与数学中复数的概念一致。定义j=根号下（-1），以此为基础，构建数学体系。

-a+bJ被称为复数，其中，a是实部，b是虚部。

例如：z=1.23e-4+5.6e+89j 的实部和虚部是什么？

用z.real获得实部，z.imag获得虚部

2、数值运算操作符

操作符是完成运算的一种符号体系。

|  |  |
| --- | --- |
| 操作符及使用 | 描述 |
| x+y | 加 |
| x-y | 减 |
| x\*y | 乘 |
| x/y | 除，x与y之商，10/3结果是3.33333333333333335 |
| x//y | 整数除，x与y之整数商，10//3结果是3 |
| +x | x本身 |
| -y | y的负值 |
| x%y | 余数，模运算 10%3结果是1 |
| x\*\*y | 幂运算，x的y次幂， |
| 当y是小数时，开方运算 10\*\*0.5结果是根号10 |

二元操作符有对应的增强赋值操作符。

|  |  |
| --- | --- |
| 增强操作符及使用 | 描述 |
| x op =y | 即x=x op y，其中op为二元操作符 |
| x+=y x-=y x\*=y x/=y  x//y x%=y x\*\*=y |
| >>> x=3.1415  >>> x \*\*=3 # 与x=x\*\*3等价  31.006276662836743 |

数字类型的关系：类型间可进行混合运算，生成结果为“最宽”类型

三种类型存在一种逐渐“扩展”或“变宽”的关系： 整数->浮点数->复数

例如：123+4.0=127.0 （整数+浮点数=浮点数）

3、数值运算符

一些以函数形式提供的数值运算功能

|  |  |
| --- | --- |
| 函数及使用 | 描述 |
| abs(x) | 绝对值，x的绝对值  abs（-10.01）结果为10.01 |
| divmod（x，y） | 商余，（x//y，x%y），同时输出商和余数  divmod（10，3）结果为（3，1） |
| pow（x,y[,z]） | 幂余，（x\*\*y）%z，[..]表示参数z可省略  pow（3，pow（3，99），10000）结果为4587 |
| round(x[d]) | 四舍五入，d是保留小数位数，默认值为0 |
| max(x1,x2,…,xn) | 最大值 |
| min（x1,x2,…,xn） | 最小值 |
| int(x) | 将x变成整数，舍弃小数部分 |
| float(x) | 将x变成浮点数，增加小数部分  如 float（4）结果为4.0 |
| complex（x） | 将x变成复数，增加虚数部分  如 complex（4）结果为4+0j |

1. 实例

天天向上的力量

基本问题：持续的价值

1. 一年365天，每天进步1%，累计进步多少呢？1.01的365次方
2. 一年365天，每天退步1%，累计剩下多少呢？0.99的365次方

问题1：千分之一的力量

#DayDayup.py

dayup=pow(1.001,365)

daydown=pow(0.999,365)

print("向上：{:.2f},向下:{:.2f}".format(dayup,daydown))

问题二：千分之五5%。和1%的力量

1. 一年365天，每天进步5%。或1%，累计进步多少呢？1.005的365次方 1.01的365次方
2. 一年365天，每天退步5%。或1%，累计剩下多少呢？0.995的365次方 0.99的365次方

#DayDay2.py

dayfactor=0.005

dayup=pow(1+dayfactor,365)

daydown=pow(1-dayfactor,365)

print("向上:{:.2f},向下:{:.2f}".format(dayup,daydown))

问题三：工作日的力量

1. 一年365天，一周5个工作日，每天进步1%
2. 一年365天，一周2个工作日，每天退步1%

这种工作日的力量，如何呢？

#DayDayup3.py

dayup=1.0

dayfactor=0.01

for i in range(365):

if i%7 in [6,0]:

dayup=dayup\*(1-dayfactor)

else:

dayup=dayup\*(1+dayfactor)

print("工作日的力量：{:.2f}".format(dayup))

问题四：工作日的努力

工作模式要努力到什么水平，才能与每天努力1%一样？

A君：一年365天，每天进步1%，不停歇

B君：一年365天，每周工作5天，休息日下降1%，要多努力呢？

#DayDayup4.py

'''根据df参数计算工作日力量的函数参数不同，这段代码可共用'''

#def保留字用于定义函数

#参数df，是一个占位符

def dayup(df):

dayup=1

for i in range(365):

if i % 7 in[6,0]:

dayup=dayup \* (1-0.01)

else:

dayup=dayup \* (1+df)

return dayup

dayfactor=0.01

#while保留字判断条件是否成立，条件成立时循环执行

while dayup(dayfactor) < 37.78:

dayfactor +=0.001

print("工作日的努力参数是：{:.3f}".format(dayfactor))

5、字符串类型及操作

字符串：由0个或多个字符组成的有序字符序列。

字符串是字符的有序序列，可以对其中的字符进行索引。索引是从0开始的。

字符串有 2类共4种 表示方法

1. 由一对单引号或双引号表示，仅表示单行字符串。
2. 由一对三引号或三双引号表示，可表示多行字符串。

字符串的序号：正向递增序号 和 反向递减序号。

字符串的使用：

使用[ ]获取字符串中一个或多个字符。

索引：返回字符串中单个字符。 <字符串>[M]

字符串切片高级用法：

使用[M:N:K]根据步长对字符串切片: <字符串>[M:N],M缺失表示至开头，N缺失表示至结尾

字符串的特殊字符：

转义符\: 转义符表达特定字符的本意。

转义符形成一些组合，表达一些不可打印的含义

字符串的操作符：

|  |  |
| --- | --- |
| 操作符及使用 | 描述 |
| x + y | 连接两个字符串x和y |
| n \* x 或 x \* n | 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 |

例1：

获取星期字符串：

#weeknameprintV1.py

weekStr="星期一星期二星期三星期四星期五星期六星期日"

weekId=eval(input("请输入星期数字(1-7):"))

pos=(weekId-1)\*3

print(weekStr[pos:pos+3])

例2：

#weekNamePrintV2.py

weekStr="一二三四五六日"

weekId=eval(input("请输入星期数字(1-7):"))

print("星期"+weekStr[weekId-1])

字符串处理函数：

一些以函数形式提供的字符串处理功能

|  |  |
| --- | --- |
| 函数及使用 | 描述 |
| len(x) | 长度，返回字符串x的长度.  例len(“一二三456”)结果为6 |
| str(x) | 任意类型x所对应的字符串形式  str(1.23)结果为“1.23” |
| hex(x)或oct(x) | 整数x的十六进制或八进制小写形式字符串  hex(425)结果为“0x1a9” oct(425)结果为“0o651” |
| chr(u) | u为Unicode编码，返回其对应字符 |
| ord(x) | x为字符，返回其对应的Unicode编码 |

Unicode编码

Python字符串的编码方式

统一字符编码，即覆盖几乎所有字符的编码方式。

从0到1114111（0x10FFFF）空间，每个编码对应一个字符。

Python字符串中每个字符都是Unicode编码字符。

例：

9801金牛

输出十二星座代码：

for i in range(12):

print(chr(9800+i),end="")

6、字符串处理方法：

“方法“在编程中是一个专有名词。

“方法’特指<a>.<b>()风格中的函数<b>()

方法本身也是函数，但与<a>有关，<a>.<b>()风格使用字符串及变量也是<a>,存在一些方法

|  |  |
| --- | --- |
| 方法及使用 | 描述 |
| str.lower()或str.upper() | 返回字符串的副本，全部字符大写或小s |
| str.split(sep=None) | 返回一个列表，由str根据sep被分隔的部分组成  “A,B,C”.split(“,”)结果为[‘A’,’B’,’C’] |
| str.count(sub) | 返回子串sub在str中出现的次数 |
| str.replace(old,new) | 返回字符串str副本，所有old子串被替换为new |
| str.center(width[,fillchar]) | 字符串str根据宽度width居中，fillchar可选  “python”.center(20,”=”)结果为‘=======python=======’ |
| str.strip(chars) | 从str中去掉在其左侧和右侧chars中列出的字符  “= python= ”.strip(“ =np”)结果为“ytho” |
| Str.join(iter) | 在iter变量除最后元素外每个元素后增加一个str  “,”.join(“12345”)结果为“1，2，3，4，5”  主要于字符串的分隔 |

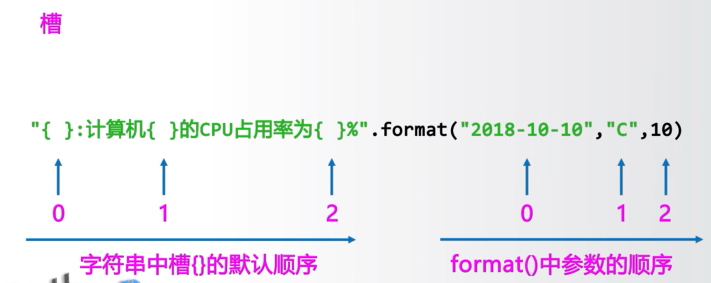
7、字符串类型的格式化

格式化是对字符串进行格式表达的方式。

字符串格式化使用.format()方法，用法：<模板字符串>.format(<逗号分割的参数>)

槽

“{ }：计算机{ }的CPU占用率为{ }%”.format(“2018-10-10”,”C”,10)



槽内部对格式化的配置方式

{<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 | 槽设定的输出宽度 | 数字的千位分隔符 | 浮点数小数  精度或字符串最大输出长度 | 整数类型  b,c,d,o,x,X  浮点数类型  e,E,f.% |

8、模块2：time库的使用

time库是python中处理时间的标准库

计算机时间的表达

提供获取系统时间并格式化输出功能

import time

time.<b>()

time库包括三类函数：

1. 时间获取：time() ctime() gmtime()
2. 时间格式化： strftime() strptime()
3. 程序计时：sleep(), perf\_counter()

时间获取：

|  |  |
| --- | --- |
| 函数 | 描述 |
| time() | 获取当前时间戳，即计算机内部时间值，浮点数 |
| ctime() | 获取当前时间并以易读方式表示，返回字符串 |
| gmtime() | 获取当前时间，表示为计算机可处理的时间格式 |

时间戳从1970年1月1日零点开始到现在。

时间格式化：

将时间以合理的方式展示出来。

格式化：类似于字符串格式化，需要有展示模板

展示模板由特定的格式化控制符组成.

|  |  |
| --- | --- |
| 函数 | 描述 |
| strftime(tpl,ts) | tpl是格式化模板字符串，用来定义输出效果；ts是计算机内部时间类型变量 |
| strptime(str,tpl) | str是字符串形式的时间值  tpl是格式化模板字符串，用来定义输入效果 |

|  |  |  |
| --- | --- | --- |
| 格式化字符串 | 日期/时间说明 | 值范围和实例 |
| %Y | 年份 | 0000~9999，例如：1900 |
| %m | 月份 | 01~12 |
| %B | 月份名称 | January-Dece |
| %b | 月份名称缩写 | Jan~Dec |
| %d | 日期 | 01~31 |
| %A | 星期 | Monday~Sunday |
| %a | 星期缩写 | Mon~Sun |
| %H | 小时（24h制） | 00~23 |
| %I | 小时（12h制） | 01~12 |
| %p | 上/下午 | AM，PM |
| %M | 分钟 | 00~59 |
| %S | 秒 | 00~59 |

程序计时：

指测量起止动作所经历时间的过程

测量时间：perf\_counter()

产生时间：sleep()

|  |  |
| --- | --- |
| 函数 | 描述 |
| perf\_counter() | 返回一个CPU级别的精确时间计数值，单位为秒 由于这个计数值起点不确定，连续调用差值才有意义 |
| sleep(s) | S拟休眠的时间，单位是秒，可以是浮点数 |

实例：文本进度条

需求分析：

采用字符串方式打印可以动态变化的文本进度条

进度条需要能在一行中逐渐变化

简单的开始：

#jindutiao.py

import time

scale=10

print("-------执行开始------")

for i in range(scale+1):

a='\*' \* i

b='.' \* (scale - i)

c=(i/scale)\*100

print("{:3.0f}%[{}->{}]".format(c,b,a))

time.sleep(0.1)

print("------执行结果-------")

单行动态刷新：

1、刷新的本质：用后打印的字符覆盖之前的字符

2、不能换行：print()需要被控制

3、要能回退：打印后光标退回到之前的位置\r

#danhangdongtaishuaxin.py

import time

for i in range(101):

print("\r{:3f}%".format(i),end="")

time.sleep(0.1)

文本进度条实例完整效果

#jindutiaowamzhengban.py

import time

scale=50

print("执行开始".center(scale//2,"-"))

start=time.perf\_counter()

for i in range(scale + 1):

a='\*' \* i

b='.' \* (scale - i)

c=(i/scale)\*100

dur=time.perf\_counter() - start

print("\r{:^3.0f}%[{}->{}]{:.2f}s".format(c,a,b,dur),end='')

time.sleep(0.1)

print("\n"+"执行结果".center(scale//2,'-'))