第三章：基本数据类型

3.1 数字类型及操作

1.整数类型

（1）pow(x,y)函数，计算 x 的 y 次方，想算多大算多大。

（2）4 种进制表示形式：

①十进制：1010,99 等；

②二进制：以 0b 或者 0B 开头：0b10,-0B101 等；

③八进制：以 0o 或 0O 开头：0o123,-0O456 等；

④十六进制：以 0x 或者 0X 开头：0x9a,-0X89 等。

2.浮点数类型

取值范围为-10 的 308 次方到 10 的 308 次方，精度数量级 10 的-16 次方。

浮点数间存在不确定尾数，不是 bug。为了解决这一个问题，可以使用 roud(x,d)函数。rand(x,d)：表示对 x 四舍五入，d 是小数截取尾位数。

Python 提供使用字母 e 或者 E 作为幂的符号，以 10 为基数。

3.复数类型

实例：

z=1.24e-4+5.6e+89j

z.real 获得实部，z.imag 获得虚部。

4.数值运算操作符

表 1 基本操作符

操作符及使用	描述
x+y	加
x-y	减
x*y	乘
x/y	除，x 除以 y，10/3=3.33333333333335
x//y	整数除，例如 10//3=3
+x	x 本身
x%y	取余数
-x	x 的负值
x**y	若 y 是整数，则是 x 的 y 次幂
	若 y 是小数，则是开方运算

注意：

类型间可进行混合运算，生成结果为“最宽”类型（参与运算元素中最宽的类型）。

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

5.数值运算函数

表 2 常用数值运算函数

函数及使用	描述

abs(x)	对 x 求绝对值
divmod(x,y)	商余，即同时输出商和余数
pow(x,y[,z])	幂余，[]表示可省略。这个式子的意思是：(x**y)%z
round(x[,d])	表示对 x 四舍五入，d 是小数截取尾位数
max(a,b,c,…)	返回 a,b,c,…中的最大值
min(a,b,c,…)	返回 a,b,c,…中的最小值

表 3 数据类型转换函数

函数及使用	描述
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。例如 float(12)结果为 12.0
complex(x)	转换成复数，增加虚部

3.2 实例：天天向上的力量

1.问题分析

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？

（2）一年 365 天，每天退步 1‰，累计退步多少呢？

（3）如果是“三天打鱼两天晒网”呢？

（4）如果是“双休日休息又不退步”呢？

2.“天天向上的力量”第一问

代码：

```
#DayDayUpQ1.py 进步或者退步 1‰
dayup = pow(1.001,365)
daydown = pow(0.999,365)
print("向上：{:.2f}，向下：{:.2f}".format(dayup,daydown))
```

结果：

```
向上：1.44，向下：0.69
```

3. “天天向上的力量”第二问

代码：

```
#进步或者退步 5‰
#DayDayUpQ2.py
dayfactor = 0.005
days = 365
dayup = pow(1+dayfactor,days)
daydown = pow(1-dayfactor,days)
print("向上：{:.2f}，向下：{:.2f}".format(dayup,daydown))
```

结果：

```
向上：6.17，向下：0.16
```

4. “天天向上的力量”第三问

一年 365 天，一周 5 个工作日，每天进步 1%；一周 2 个休息日，每天退步 1%，这种工作日

的力量又如何呢？

代码：

```
#DayDayUpQ3.py
dayfactor = 0.01
dayup =1.0
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))
```

运行结果：

```
工作日的力量：4.63
```

5. “天天向上的力量”第四问

问题 4：工作日模式（即平时努力工作，双休每天下降 1%）要努力到什么水平，才能与每天都进步 1%取得的结果一样呢？

流程图：

图 1

代码：

```
#定义函数
def dayUP(df):
    dayup = 1
    for i in range(365):
        if i%7 in [0,6]:
            dayup = dayup*(1-0.01)
        else:
            dayup = dayup*(1+df)
    return dayup
dayfactor = 0.01
dayupa1 = pow(1.01,365)
dayupa = round(dayupa1,2)
while dayUP(dayfactor) <37.78:
    dayfactor += 0.001
print("工作日努力的参数是：{:.3f}".format(dayfactor))
```

运行结果：

```
工作日努力的参数是：0.019
```

3.3 字符串类型及操作

1.字符串类型的表示

字符串是由 0 个或多个字符组成的有序字符序列。字符串由一对单引号或者双引号表示。字符串是字符的有序序列，可以对其中的字符进行索引。

字符串由 2 类共 4 中表示方法：

（1）由一对单引号或者双引号表示；

（2）一对三单引号或三双引号表示，可表示多行字符串。**当所给出的多行字符串没有被应用，则就当做了注释。**

字符串切片的高级用法

使用[M:N:K]根据步长对字符串进行切片，M 缺失表示至开头，N 缺失表示到结尾，K 表示步长。例如：“〇一二三四五六七八九十”，[1:8:2]的结果是“一三五七”。[::-1]则是“十九八七六五四三二一〇”。

2.字符串操作符

表 4 字符串操作符

函数及使用	描述
x+y	连接两个字符串 x 和 y
n*x 或者 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

3.字符串处理函数

表 5 常用字符串处理函数

函数及使用	描述
len(x)	返回字符串 x 的长度
str(x)	任意类型 x 所对应字符串格式。如 str([1,2])结果为”[1,2]”
hex(x)或 oct(x)	整数 x 的十六进制或八进制的字符串形式
chr(x)	x 为 Unicode 编码，返回其对应的字符
ord(x)	x 为字符，返回其对应的 Unicode 编码

Python 字符串中每个字符都是 Unicode 编码字符，从 0 到 1114111（0x10FFFF）空间，每个编码对应一个字符。

例如：

```
for i in range(12):
    print(chr(9800+i),end = "")
#如果 end 后面为空，则不换行
```

结果：

♈♉♊♋♌♍♎♏♐♑♒♓

4.字符串处理方法

此处方法特指<a>.<b>()中的函数<b>()，方法本身也是函数，但与<a>有关。

表 6 常用字符串处理方法

函数及使用	描述
str.lower()或 sr.upper()	返回字符串的副本，全部小写/大写
str.split(sep=None)	返回一个列表，由 str 根据 sep 被分割的部分组成。例如，”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，所有的 old 子串被替换为 new
str.center(width[,fillchar])	字符串 str 根据宽度 width 剧中，fillchar 可选。例如”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。例

	如，”= python=”.strip(“ =np”)结果为”ytho”
str.jpin(iter)	在 iter 变量除最后元素外每个元素后面增加一个 str。例如，”,”.join(“12345”结果为”1,2,3,4,5”)

5.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。用法如下：<模板字符串>.format(<逗号分隔的参数>)

我们需要用到一个概念——槽。相当于一个占位符，通常用{}表示。例如：

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

输出为：

2018-10-10:计算机 C 的 CPU 占用率为 10%

若代码为：

“{1}:计算机{0}的 CPU 占用率为{2}%”.format(“2018-10-10”,”C”,10)

则输出为：

C:计算机 2018-10-10 的 CPU 占用率为 10%

槽内部对格式化的配置方式：{<参数序号>:<格式控制标记>}，具体内容如图 2 所示：

图 2

对于前三类举例：

“{0:=^20}”.format(“PYTHON”)

在这里面，“：”冒号是一个引导符，前面的数字 0 代表的是用第 0 个元素来放到这里，“=”等号代表用于填充的字符，^表示居中对齐，20 代表槽的输出宽度。

对于后三类举例：

“{0:,.2f}”.format(12345.6789)

在这里，0 还是指填充第 0 个元素到这里，“：”还是引导符，“,”是数字的千位分隔符，后面的 2 是小数精度，f 是数据类型。输出结果是：12,345.68（7 四舍五入进 1）

3.4 模块 2：time 块的应用

1.time 库基本介绍

time 库是 Python 处理时间的标准库，能够提供计算机时间的表达，提供获取系统事件并格式化输出功能，提供系统级精确计时功能，用于程序性分析。

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()

2.时间获取

表 7 time 库的函数

函数	描述
time()	获取当前时间戳，即计算机内部是兼职，浮点数（相对于 1976-1-1 的秒数）例如：1522157003.0202858
ctime()	获取当前时间并以易读方式表示，返回字符串。例如：Tue Mar 27 21:23:23 2018

gmtime()	获取当前时间，表示为计算机可处理的时间格式。例如：time.struct_time(tm_year=2018，tm_mon=3，tm_mday=27，tm_hour=13，tm_min=23，tm_sec=23，tm_wday=1，tm_yday=86, tm_isdst=0)

3.时间格式化

它是将时间以合理的方式展示出来，它也需要一个展示模板，由特定的格式化控制符组成。

表 8 time 时间格式化的函数

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 是计算机内部事件类型变量。例如运行： t = time.gmtime() print(time.strftime("%y-%m-%d %H:%M:%S",t))结果为：18-03-27 13:27:30
strptime(str,tpl)	将一段字符串 str 按照 tpl 的格式来编程时间值。

表 9 时间格式化字符串

格式化字符串	日期/时间说明	值范围和实例
%Y	年份	0000~9999
%m	月份	01~12,
%B	月份名称	January~December
%b	月份名称缩写	
%d	日期	01~31
%A	星期	Monday~Sunday
%a	星期缩写	
%H	小时（24 小时制）	00~23
%h	小时（12 小时制）	
%p	上午/下午	AM/PM
%M	分钟	00~59
%S	秒	00~59

4.程序计时应用

测试时间：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。

产生时间：sleep(s)：让程序休眠 s 秒，可以是浮点数。

3.5 实例：文本进度条

1.问题分析

采用字符串方式打印可以动态打出的文本进度条。采用 sleep()模拟一个持续的进度。

2 简单的开始

代码：

```
scale = 10
for i in range(scale-1):
    a = '*'*i
    b = '.'*(scale - i)
    c = (i/scale)*100
    print("{:^3.0f}%[{}->{}]".format(c,a,b))
    time.sleep(0.1)
print("------FINISHED------")
```

结果：

```
0 %[->.........]
10 %[*->........]
20 %[**->.......]
30 %[***->......]
40 %[****->.....]
50 %[*****->....]
60 %[******->...]
70 %[*******->..]
80 %[********->.]
------FINISHED------
```

3.单行动态刷新

刷新的本质是：用后打印的字符覆盖之前的字符；

不能换行：print()需要被控制

要能回退：打印后光标退回到之前的位置\r

代码：

```
import time
for i in range(101):
    print("\r{:3}%".format(i), end = "")
#想在输出的字符串后面加啥就是 end 里面加啥，不换行就是空
    time.sleep(0.1)
```

4.实例完整效果

代码：

```
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("\nFINISHED".center(scale//2,'.'))
```