

Python 程序设计-2023

张建章

阿里巴巴商学院

杭州师范大学

2023-12



目录 |

- 1 课程考核说明**
- 2 关于课程**
- 3 计算思维**
- 4 如何学好程序设计**
- 5 配置本机环境**
- 6 jupyter-lab 常见问题与解答**
- 7 Python 下载与安装**
- 8 两种代码执行方式**
- 9 基本数据类型: 整数**
- 10 基本数据类型: 浮点数**
- 11 基本数据类型: 复数**
- 12 基本数据类型: 字符串**
- 13 基本数据类型: 布尔值**
- 14 基本数据类型: 空值**
- 15 变量**

目录 II

- 16 操作符
- 17 语句和表达式
- 18 函数
- 19 简单输入与输出
- 20 模块
- 21 简单条件判断语句
- 22 简单循环语句
- 23 写程序注意事项
- 24 Python 常见的内置数据类型
- 25 列表
- 26 列表的基本操作
- 27 常用的操作列表的内置函数
- 28 常用的列表方法
- 29 内置函数与列表方法的区别
- 30 多维列表

目录 III

- 31 元组
- 32 元组封装与序列拆封
- 33 元组与列表的比较
- 34 字符串操作
- 35 字符串常用方法
- 36 常用的处理字符串的内置函数
- 37 转义字符
- 38 字符串格式化
- 39 字符串格式化的功能性
- 40 字符串格式化的装饰性
- 41 集合
- 42 集合的基本操作
- 43 字典
- 44 字典的基本操作
- 45 字典的常用内置方法

目录 IV

- 46 深复制和浅复制
- 47 Python 中的自助函数
- 48 三种基本控制结构
- 49 选择结构
- 50 循环
- 51 break,continue,pass, 嵌套
- 52 循环结构与 else 子句搭配
- 53 课堂实训
- 54 课后练习
- 55 定义函数
- 56 变量的作用域
- 57 函数的参数
- 58 函数实例
- 59 匿名函数
- 60 map、reduce、filter 函数

目录 V

- 61 递归函数
- 62 生成器
- 63 open 函数
- 64 文件内容读取
- 65 文件内容写入
- 66 文件内移动游标
- 67 文件读写实例
- 68 文件读写工具包
- 69 模块与包的定义
- 70 包管理
- 71 模块与包的使用
- 72 异常类型
- 73 异常处理
- 74 断言

根据教学大纲要求，本课程的考核办法为：

$$\begin{aligned}\text{总成绩} = & \text{期末成绩} \times 50\% + \text{日常作业} \times 30\% \\ & + \text{日常考勤} \times 10\% + \text{课堂表现} \times 10\%\end{aligned}$$

其中，期末考试采用上机考试形式。

课程名称：《Python 程序设计》

课程目标：

- ① 掌握 Python 编程语言；
- ② 培养“计算思维”；
- ③ 通过程序设计高效解决实际问题。

授课方式：上机实验为主，主要基于 Jupyter-lab 交互式编程教学

作业提交：坚果云在线提交 Jupyter Notebook 文件 (后缀为.ipynb)

商科同学为什么要学 **Python**——赋能数字经济研究与实践：

2. 关于课程



数据分析



可视化



舆情分析



量化投资

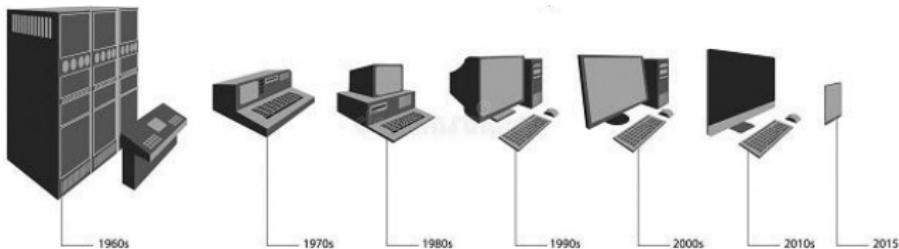
计算思维 (computational thinking): 计算机科学家在用计算机解决问题时特有的思维方式和解决方法。

- 基本原则：既充分利用计算机的计算和存储能力，又不超出计算机的能力范围。
- 不同阶段：问题表示 → 算法设计 → 编程技术 → 可计算性与算法复杂性。
- 生活实例：菜谱中的“勾芡”类似模块化，厨师同时做多个菜类似并发，书包中的书类似缓冲存储。

拓展阅读：如何理解计算思维

计算机与计算机语言

计算机是一种能够按照事先存储的程序(可编程性),自动、高速地对数据进行输入、处理、输出和存储的系统(功能性)。



计算机语言

- 用于人与计算机之间交互的语言;
- 一套用以编写计算机程序的数字、字符和语法规则;
- 是计算机程序的实现方式;
- 计算机语言比自然语言更为简洁、精确和严谨。

编译和解释

源代码是采用某种编程语言编写的计算机程序，人类可读，如 C、Java、Python 等。

目标代码是计算机可直接执行的代码，人类不可读，如 class 文件。

编译指编译器将代码一次性转换成目标代码的过程。

解释指解释器将源代码逐条转换成目标代码并逐条运行的过程。



编译



解释

静态语言和脚本语言

静态语言

- 使用编译执行的编程语言，如 Java；
- 一次性生成目标代码，优化充分，执行效率高。



脚本语言

- 使用解释执行的编程语言，如 Python、Ruby；
- 简化了“开发、部署、测试和调试”的周期过程。



计算机编程的基本原则

- 精确无歧义；
- 计算机只能按照给定的指令一步步做，无跳跃（机械执行）；
- 按照计算机的特点去思考，计算思维；
- 充分考虑计算机的能力和限制之上；



计算机编程的基本方法

- ① 输入 (Input): 终端命令行交互, 文件, 网络;
- ② 处理 (Processing): 对输入数据进行计算并产生输出结果的过程;
- ③ 输出 (Output): 通过终端命令行, 文件, 网络等输出结果。



计算实例：体质指数

体质指数 (Body Mass Index, BMI) = $\frac{\text{体重 (kg)}}{\text{身高 (m)}^2}$ 。

BMI 值是一个中立而可靠的指标，是国际上常用的衡量人体胖瘦程度以及是否健康的一个标准。

计算机：

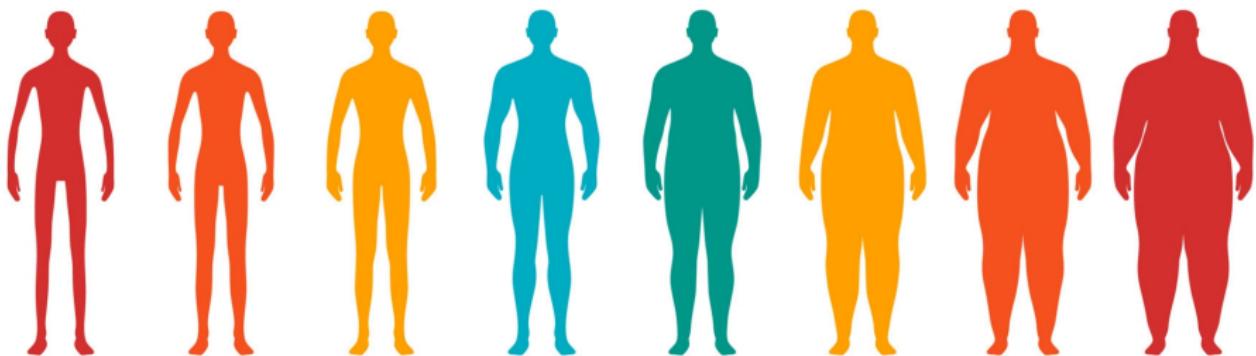
- ① 接收用户输入的身高、体重数据；
- ② 根据公式计算；
- ③ 输出体质指数。

用户：

- ① 输入身高、体重数据；
- ② 查看计算结果，对照量表；

BMI 对照表

BODY MASS INDEX (kg/m²)



< 16	16 - 17	17 - 18.5	18.5 - 25	25 - 30	30 - 35	35 - 40	> 40	
Severe Thinness	Moderate Thinness	Mild Thinness	Normal		Overweight	Obese Class I	Obese Class II	Obese Class III

为什么选择 Python

- 可移植性强

开源本质，Python 已经被移植在许多平台；

- 庞大的标准库与丰富的第三方生态库 (**PyPI, Github**)

位于编程语言生态链的顶级位置；

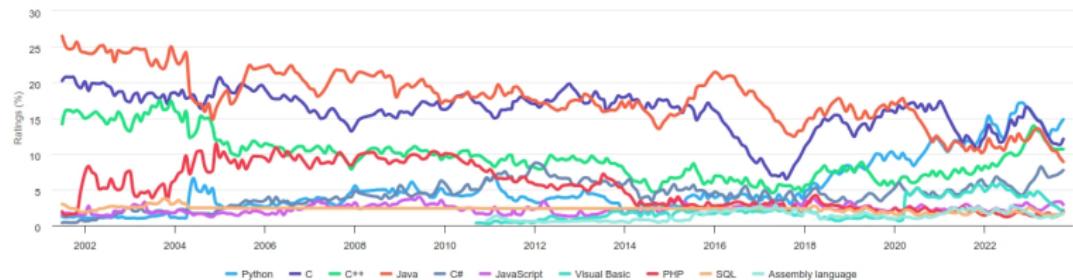
可用于网站、搜索引擎、云计算、大数据、人工智能、科学计算；

- 简洁高效

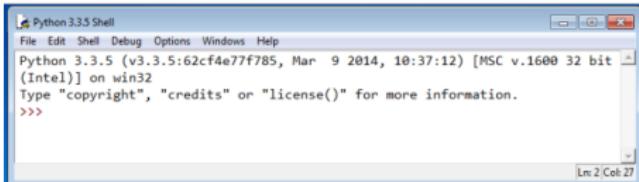
代码量小，开发调试效率高。



Python 语言的流行度高



让 Python 编程更高效的工具



A screenshot of the Python 3.3.5 Shell window. The menu bar includes File, Edit, Shell, Debug, Options, Windows, Help. The status bar shows "Python 3.3.5 (v3.3.5:62cf4e77f785, Mar 9 2014, 10:37:12) [MSC v.1600 32 bit (Intel)] on win32". The command line shows "Type "copyright", "credits" or "license()" for more information." and a prompt "=>". The bottom status bar indicates "Ln 2 Col 27".

原生 Python 和 IDE



能跑的车架子



集成开发环境



跑车

为了学 Python 我需要什么样的电脑



没有必要购买高端电脑，如外星人



市面上普通的电脑即可用于本课程学习

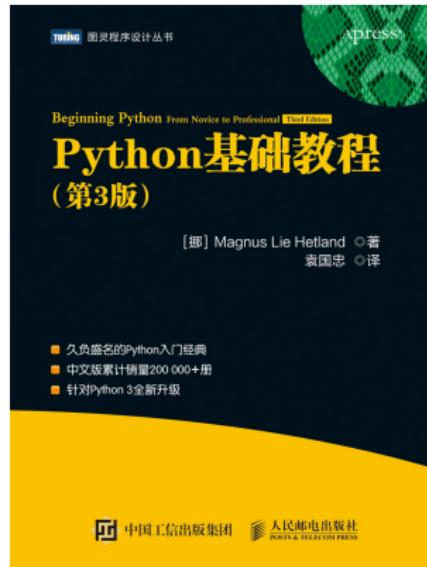
学习资源



《程序设计基础》课程网站

课程讨论区 

课堂和课后互动



参照课本勤奋练习



善于搜索互联网

Windows 安装 Anaconda

判断 32 位和 64 位 Windows



在线视频

Jupyter-lab 基本用法



[在线视频, 文字说明 \(For Windows\)](#); [在线视频, 文字说明 \(For Mac\)](#)

用 Python 计算 BMI

```
Height = float(input("请输入身高(m): "))
Weight = float(input("请输入体重(kg): "))

BMI = round(Weight/Height**2, 2)

if BMI>=23.9:
    print("BMI指数为", BMI, "体质偏重")
elif BMI<=18.5:
    print("BMI指数为", BMI, "体质偏轻")
else:
    print("BMI指数为", BMI, "正常")
```

(1) jupyter-lab 启动后无法成功运行

- ① 在使用 jupyter-lab 的过程中，确保 cmd 黑框框处于打开状态；
- ② 如果 cmd 黑框框显示 **Bad File Descriptor** 错误，类似下图

```
To access the notebook, open this file in a browser:  
    file:///C:/Users/%E5%B8%85%E5%B8%85%E9%A3%9E%E7%8C%AA/AppData/Roaming/jupyter/lab/notebooks/Untitled.ipynb  
Or copy and paste one of these URLs:  
    http://localhost:8888/?token=47cf2aaa44780278c4e644e8c277c5088e44a5cca0  
    or http://127.0.0.1:8888/?token=47cf2aaa44780278c4e644e8c277c5088e44a5cca0  
[I 18:56:16.832 NotebookApp] 302 GET / (::1) 0.000000ms  
[W 18:56:44.320 NotebookApp] 404 GET /nbextensions/widgets/notebook/js/extensio  
00ms referer=http://localhost:8888/notebooks/TPPractise/Untitled.ipynb  
Bad file descriptor (C:\projects\libzmq\src\epoll.cpp:100)  
Bad file descriptor (C:\projects\libzmq\src\epoll.cpp:100)
```

解决办法：关闭当前 cmd 黑框框，重新打开 cmd 黑框框，在确保网络连通的情况下，依次执行如下两条命令 `pip uninstall pyzmq`；
`pip install pyzmq==19.0.2 --user`，两条命令成功运行后（运行时没有出现 Error 信息），重新启动 jupyter-lab 即可。

(2) jupyter-lab 的浏览器界面需要输入 token

请复制你 cmd 黑框框中的 http 开头的网址 (两个网址中的任意一个, 类似下图) 到浏览器打开。

```
To access the notebook, open this file in a browser:  
file:///C:/Users/Administrator/AppData/Roaming/jupyter/runtime/nbserver-58048-open.html  
Or copy and paste one of these URLs:  
http://localhost:8888/?token=adda914e1a9f67dc96d78601ae42e1dae7cb10efa3bc06b3  
or http://127.0.0.1:8888/?token=adda914e1a9f67dc96d78601ae42e1dae7cb10efa3bc06b3  
[W 23:49:14.572 LabApp] Could not determine jupyterlab build status without nodejs  
[I 23:49:16.360 LabApp] Kernel started: bde3aa9e-ea4a-4fa0-a89c-e46129c63b26
```

(3) 运行 BMI_calculation 代码时, 输入身高体重后无法继续计算

确保在输入身高体重信息后, 按 Enter 键确认, 因为 input 函数在接受键盘输入后, 需要用户确认输入, 以继续运行后续程序代码。

(4) 明明在 jupyter-lab 里写了代码, 却在本机上找不到

请在启动 jupyter-lab 后, 进入到桌面 Desktop, 然后新建 Notebook, 重命名为有意义的英文名字, 再写代码, 写代码过程中, 一定要多按保存键, 快捷键为 Ctrl + S。

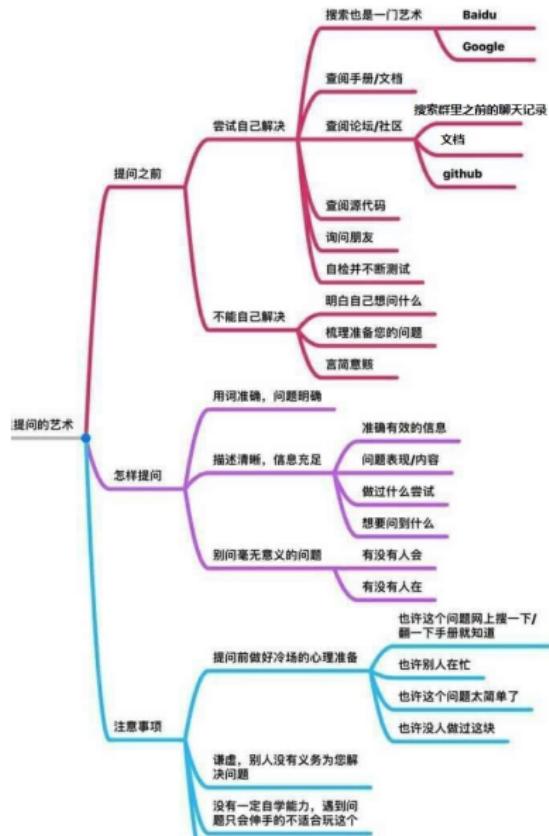
(5) 课程配套代码的打开与运行

本地：①将课程网站的代码下载到本机，建议下载到桌面，便于查找；②启动 jupyter-lab，从左侧文件导航栏找到本地保存的课程代码，双击后打开；③运行选中的当前代码行，即当前选中的 cell，点击形如播放的按钮，运行全部代码，点击形如快进的按钮。

云端：①将课程网站的代码下载到本机，建议下载到桌面，便于查找；②打开魔搭在线 jupyter 环境，左侧文件导航栏上方，点击向上的箭头，上传课程代码到云端；③启动 jupyter-lab，从左侧文件导航栏找到本地保存的课程代码，双击后打开；④运行选中的当前代码行，即当前选中的 cell，点击形如播放的按钮，运行全部代码，点击形如快进的按钮。

注意：一定要自己练习课程提供的代码，切忌只看代码，而不动手写代码和运行自己写的代码。

(6) 提问的艺术



安装 Python 3.X 并启动: 本课程安装的 Anaconda 3.X 已内含 Python 3.X, 启动 Jupyter-lab 即启动 Python;



Python 采用编译/解释混合方式: 先编译成字节码, 再解释执行。

8. 两种代码执行方式

交互式

```
[1]: # 计算BMI指数
[2]: Height = float(input("请输入身高(m): "))
请输入身高(m): 1.75
[3]: Weight = float(input("请输入体重(kg): "))
请输入体重(kg): 62
[4]: BMI = round(Weight/Height**2, 2)
[5]: if BMI>=23.9:
    print("BMI指数为", BMI, "体质偏重")
elif BMI<=18.5:
    print("BMI指数为", BMI, "体质偏轻")
else:
    print("BMI指数为", BMI, "正常")
BMI指数为 20.24 正常
```

脚本式

```
zjz@dell: ~
(base) zjz@dell:~$ python BMI_calculation.py
请输入身高(m): 1.75
请输入体重(kg): 62
BMI指数为 20.24 正常
(base) zjz@dell:~$ |
```

Python 支持多种进制类型

二进制 (Binary, 0 ~ 1), 以 `0b` 或者 `0B` 为前缀;

八进制 (Octal, 0 ~ 7), 以 `0o` 或者 `0O` 为前缀;

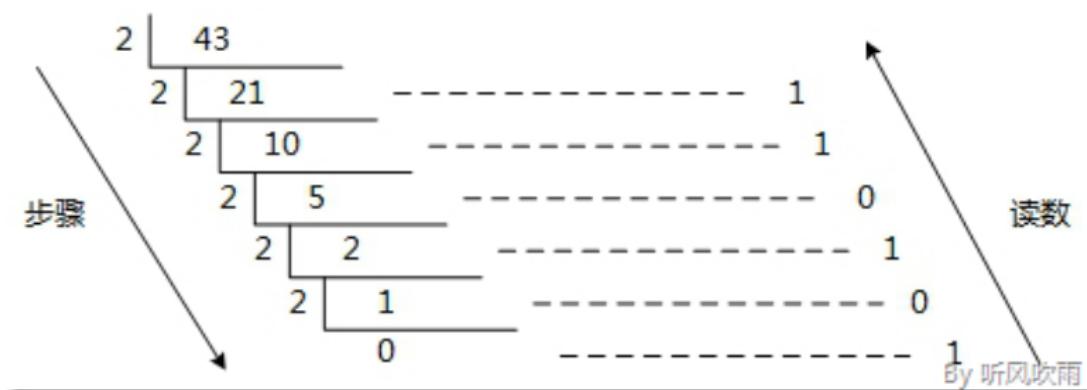
十进制 (Decimal, 0 ~ 9), Python 默认使用的进制, 不需要输入前缀;

十六进制 (Hexadecimal, 0 ~ 9, A ~ F), 以 `0x` 或者 `0X` 为前缀。

Python 3.7.X+ 可以表示任意大小整数

十进制到其他进制 (n) 的转换

除 n 取余法: 即每次将整数部分除以 n , 余数为该位权上的数, 这个步骤一直持续下去, 直到商为 0 为止, 读数时, 从最后一个余数读起。下图为十进制数 43 转化为二进制数 101011 的计算示例。

图 1: $(43)_D = (101011)_B$

其他进制 (n) 到十进制的转换

以二进制为例: 二进制数从低位到高位 (从右向左) 计算, 第 0 位的权值是 2 的 0 次方, 第 1 位的权值是 2 的 1 次方, 第 2 位的权值是 2 的 2 次方, 依次递增下去, 把最后的结果相加的值即为十进制的数值。

八进制、十六进制转十进制的方法与二进制转十进制类似。

二进制数 101011 转换为十进制数, 如下:

$$\begin{aligned}101011 &\rightarrow 1 \times 2^0 + 1 \times 2^1 + 0 \times 2^2 + 1 \times 2^3 + 0 \times 2^4 + 1 \times 2^5 \\&= 1 + 2 + 0 + 8 + 0 + 32 \\&= 43\end{aligned}$$

Python 中的进制转换函数

通过内置函数 `bin`, `oct`, `hex` 实现 10 进制转换为其他进制:

```
bin(43) # 0b101011  
oct(796) # 0o1434  
hex(796) # 0x31c
```

通过内置函数 `int` 实现其他进制转十进制:

```
int('101011',2) # 43  
int('53', 8) # 43  
int('2B', 16) # 43
```

普通计数法和科学计数法 (用 E 或 e 表示底数 10)

```
0.1233333445 # 0.1233333445  
1E-3 # 0.001
```

存在不定尾数，有些浮点数无法精确表达

```
0.1 + 0.2 # 0.3000000000000004  
0.6 + 1.2 # 1.7999999999999998
```

形如 $z = a + bi$ (a, b 均为实数) 的数称为复数。其中, a 称为实部, b 称为虚部, i 称为虚数单位。

- 实数可以被认为是虚部为零的复数, 实数 a 等价于复数 $a + 0i$;
- 实部为零且虚部不为零的复数也被称作纯虚数, 如, $2i$;
- 实部不为零且虚部也不为零的复数也被称作非纯虚数, 如 $3 + 2i$ 。

Python 中表示复数要注意: ① 用字母 j 来表示虚数单位 i ; ② 虚部为 1 时, 1 不可以省略。

```
3+2j # (3+2j)
1 + 1j # (1+1j)
```

字符串在解释器中通常高亮显示:

```
'Great Company, 好公司, gute Firma, 良い会社, Хорошая компания, شركة جيدة'
```

```
'Great Company, 好公司, gute Firma, 良い会社, Хорошая компания, شركة جيدة'
```

Python 中使用字符串需注意:

- 字符串写在引号 (单引号、双引号、三引号) 内;
- 字符串可以是空的 (一对引号中什么字符也没有);
- 字符中包含引号时: 使用转义字符, 或者, 字符串边界使用的引号与字符串中的引号不是同一类型 (如, 双引号中可以直接输入单引号作为字符串的一部分)。

13. 基本数据类型: 布尔值

布尔值有两个: `True` (1, 真), `False` (0, 假), 布尔值可以直接参与运算, `True` 相当于 1, `False` 相当于 0。

```
True == 1 # True
```

```
True - 3 # -2
```

```
False + True # 1
```

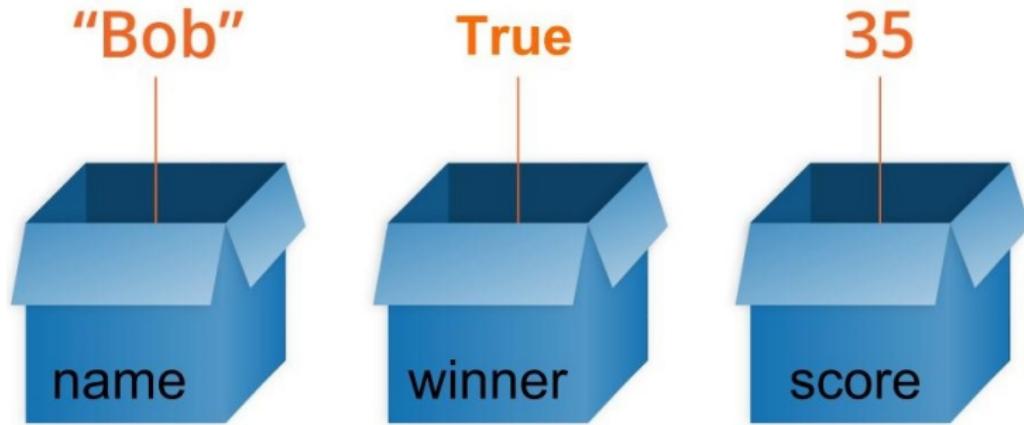
```
1 + true - false # Error, 必须首字母大写
```

空值用 `None` 表示，不能直接参与运算。

```
None + 3 # Error
```

```
False == None # False
```

变量的创建



一个数据在计算机内需要一个对应的内存空间，每个内存空间存在一个地址，通过地址程序可以访问内存中的数据，**变量名与变量地址绑定**，可以通过**变量名来访问数据**。Python 内置函数 `id` 返回变量的内存地址，用整数表示。

`university_name = 'HZNU'`， `university_name` 是变量名，
HZNU 是变量值。

变量的命名规则

- 只能由字母，数字和下划线组成；
- 不能以数字开头；
- 不能与 Python 关键字、内置函数名等 Python 内置的标识符重名；
- 大小写敏感；
- 必须要有意义 (你自己的名字有意义，变量名也要有意义哦)。

Python 关键字



```
# Python program that prints the complete list of keywords

import keyword
print(keyword.kwlist)

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

运算操作符

Operator	Meaning	Example
+	Addition	$4 + 7 \longrightarrow 11$
-	Subtraction	$12 - 5 \longrightarrow 7$
*	Multiplication	$6 * 6 \longrightarrow 36$
/	Division	$30 / 5 \longrightarrow 6$
%	Modulus	$10 \% 4 \longrightarrow 2$
//	Quotient	$18 // 5 \longrightarrow 3$
**	Exponent	$3 ** 5 \longrightarrow 243$

比较操作符

Operators	Meaning	Example	Result
<	Less than	$5 < 2$	False
>	Greater than	$5 > 2$	True
\leq	Less than or equal to	$5 \leq 2$	False
\geq	Greater than or equal to	$5 \geq 2$	True
$=$	Equal to	$5 == 2$	False
\neq	Not equal to	$5 != 2$	True

逻辑操作符

Operator	Meaning	Example	Result
and	Logical and	(5<2) and (5>3)	False
or	Logical or	(5<2) or (5>3)	True
not	Logical not	not (5<2)	True

在 Python 中，非布尔值也可以与逻辑运算符一起使用：

- 任何非零数字为 `True`；
- 零为 `False`；
- 空字符串为 `False`；
- 非空字符串为 `True`。

当使用非布尔值的逻辑运算符时，Python 中与运算找 `False`，或运算找 `True`，
`print(5 and 3) # Output: 3,`
`print('' or 'b') # Output: b`

身份和成员关系运算符

身份运算符 `is`：如果 `is` 两边的变量指向相同的数据对象（即内存地址相同）则返回 `True`，否则返回 `False`，可以使用函数 `id` 查看某个变量指向的数据对象所在的内存地址。

成员关系运算符 `in`：如果变量或值存在序列中，则返回 `True`，否则返回 `False`。

等于运算符 `==`：如果两边的值相同，则返回 `True`，否则返回 `False`。

前两个运算符均可以与 `not` 结合使用，表示否定，`is not`，`not in`。

`is` 与 `==` 的区别：双胞胎两个人长得一模一样 (`==`)，但他 (她) 俩不是同一个人 (`is`)。

运算符优先级

Precedence	Operator Sign	Operator Name
Highest	$**$	Exponentiation
	$+X, -X, \sim X$	Unary positive, unary negative, bitwise negation
	$*, /, //, \%$	Multiplication, division, floor, division, modulus
	$+, -$	Addition, subtraction
	$<<, >>$	Left-shift, right-shift
	$\&$	Bitwise AND
	\wedge	Bitwise XOR
	$ $	Bitwise OR
	$==, !=, <, \leq, >, \geq, \text{is}, \text{is not}$	Comparison, Identity
	not	Boolean NOT
	and	Boolean AND
Lowest	or	Boolean OR

运算符优先级

先执行优先级高的运算，优先级相同，则从左向右执行（左结合性），为了确保代码的可读性和正确性，在涉及多个运算符的复杂表达式中使用括号来明确指定操作的顺序：

```
x = True  
y = False  
z = True  
# 输出: True, 因为 "not y" 优先于 "and" 和 "or"  
print(x and not y or z)
```

```
a = 5  
b = 3  
c = 8  
# 输出: False, 因为 "a < b" 和 "b < c" 的优先级高于 "and"  
print(a < b and b < c)
```

语句

- 表达某事
- 结果是一个 python 执行动作
- 如, 赋值语句 `x = 1`

表达式

- 做某事
- 结果是一个值
- 如, 逻辑表达式 `x != 1`

注意: ① 交互解释器 (如 jupyter-lab, Python Shell) 会把所有表达式的值输出; ② 语句改变了事物, 但没有返回值, 也不会有输出 (上面的赋值语句改变了变量 `x` 的值, 执行后没有输出, 上面的逻辑表达式输出一个布尔值)。

赋值语句和算术表达式

赋值语句：给变量赋值的语句。

常用的增强的赋值运算符由算数运算操作符和等号组合而成(两个符号中间啥也没有哦),如`+=`,`/=`,`%=`等,`x %= 5`等价于`x = x % 5`。

算术表达式：包含各种算数运算符的计算表达式。

```
# 使用等号进行赋值
```

```
x = 5
```

```
# 使用增强的赋值语句进行赋值
```

```
x += 5 # 等价于x = x + 5
```

```
# 算术表达式
```

```
2 + 3 * 5 ** 2 % 4 # (2 + ((3 * (5 ** 2)) % 4))
```

函数：只有在被调用时才会执行的一个代码块，可以将数据以参数形式传递给函数，函数也可以返回数据作为结果，也什么都不返回。

函数调用是一种表达式，如 `print("Hello Kitty")` 就是在调用 Python 的内置函数 `print`，传递的参数为字符串 `"Hello Kitty"`

```
# 定义一个函数
def print_fan_name(fan_name):
    print("Hello Everybody, I am {0}, a fan of Jay Chou
          → (周杰伦)!".format(fan_name))
# 调用函数
fan_name = 'zjzhang'
print_fan_name(fan_name)
```

常用内置函数

- `bool` 表示转成布尔值;
- `complex` 表示转成复数, 可接收形如 '`'1+2j'`' 的字符串作为参数;
- `float`, `int` 分别表示转成浮点数或整数;
- `str` 表示转成字符串;
- `chr` 表示 ASCII 值或 Unicode 值转字符;
- `ord` 表示字符转 ASCII 值或者 Unicode 值;
- 点我查阅全部内置函数

ASCII (American Standard Code for Information Interchange, 美国信息交换标准代码) 是基于拉丁字母的一套电脑编码系统。它主要用于显示现代英语, 而其扩展版本可以部分支持其他西欧语言
([更多信息点我查看](#))。Unicode 整理、编码了世界上大部分的文字系统, 使得电脑可以用更为简单的方式来呈现和处理文字。

Python 中最常用的输入和输出函数分别为 `input` 和 `print`。

`input` 函数接收任意键盘输入作为参数，并将其转化为字符串，注意，键盘输入后要记得回车确认。

`print` 函数接收任意对象作为参数，并将其打印输出在屏幕，其 `end` 参数是可选的，只接收字符串值，表示打印的结束标志符。

```
# 无论你键盘输入任何内容, input函数都会将其转换为字符串,
```

```
→ 然后赋值给变量x
```

```
x = input('你输入点东西呗: ')
```

```
# end参数默认值为\n, 表示换行
```

```
print('你瞅啥') # 你瞅啥
```

```
print('瞅你咋地', end = '!!!!') # 瞅你咋地!!!!
```

Python 中一个模块对应一个 **.py** 文件，导入模块的语法为 **import** 模块名，亦可以访问模块中的变量或者函数，语法为 **from** 模块名 **import** 函数名/变量名

```
import math

# 访问模块中的变量-方式1
math.pi # 3.141592653589793
```

```
# 访问模块中的函数-方式1
math.sin(0.5*math.pi) # 1.0
```

```
from math import pi, sin
```

```
# 访问模块中的变量-方式2
pi # 3.141592653589793
```

```
# 访问模块中的函数-方式2
sin(0.5*math.pi) # 1.0
```

下面尝试用 **sympy** 包中的模块求解高数中的微积分计算题，下面代码分别计算函数 e^{x^2} 的导数、极限 $\lim_{x \rightarrow 0} \frac{1}{x}$ 、积分 $\int_0^\infty e^{-x} dx$ 、双重积分 $\int_{-\infty}^\infty \int_{-\infty}^\infty e^{-x^2-y^2} dx dy$ 。

```
# 导入sympy包中的全部模块
from sympy import *

# 定义变量
x, y = symbols("x y")

diff(cos(x), x) # 求导数

limit(1/x, x, 0) # 求极限

integrate(exp(-x), (x, 0, oo)) # 求积分

integrate(exp(-x**2 - y**2), (x, -oo, oo), (y, -oo, oo)) # 求积分
```

不要使用 **sympy** 包代替手算做数学习题!!!

根据条件是否成立决定执行哪种操作，如下：

```
score = float(input('你的成绩是: '))
# 判断成绩是否及格
if score >= 60 and score <= 100:
    print('及格')
else:
    print('不及格')
```

一定要记得写条件判断语句中的冒号，并且要保证代码对齐，即不同层级的代码要对齐（即，缩进的空格数量一致，jupyter-lab 等代码编辑器具有自动缩进对齐代码功能）。

for 循环

for 循环可以遍历序列中的每一个元素，并对元素进行操作，如下：

```
for letter in 'Python':      # 第一个实例
    print("当前字母: %s" % letter)

fruits = ['banana', 'apple', 'mango']
for fruit in fruits:          # 第二个实例
    print ('当前水果: %s' % fruit)

print ("Good bye!")
```

一定要记得写循环语句中的冒号，并且要保证代码对齐，即不同层级的代码要对齐（即，缩进的空格数量一致，jupyter-lab 等代码编辑器具有自动缩进对齐代码功能）。

while 循环

while 后面的条件为真时，执行循环体内的语句，格式如下：

```
a = 1
while a < 10:
    print(a)
    a += 2
```

[点我查看上述循环执行过程动图](#)

一定要记得写循环语句中的**冒号**，并且要保证代码对齐，即不同层级的代码要对齐(即，缩进的空格数量一致，`jupyter-lab` 等代码编辑器具有自动缩进对齐代码功能)。

唯手熟尔

- 避免拼错标识符，如变量名，函数，语句等
- 避免使用中文符号，如引号，逗号，括号等
- 引号、括号通常成对使用，如，有左括号也要有右括号，左边有引号，右边引号也别漏；
- 注意书写格式 (冒号，缩进，对齐)

序列: 列表 (可变), 元组 (不可变), 字符串 (不可变), 序列中的内容是有顺序的, 其正向和反向查找索引如下图所示:

The diagram illustrates a sequence of characters: G E E K S F O R G E E K S. This sequence is presented in a grid with 13 columns. Above the grid, row indices from 0 to 12 are listed. Below the grid, corresponding negative indices from -13 to -1 are listed. The characters are arranged as follows:

G	E	E	K	S	F	O	R	G	E	E	K	S
0	1	2	3	4	5	6	7	8	9	10	11	12
-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

©G

图 2: 序列的正向索引和反向索引

集合: 无重复元素;

字典: 存放具有映射关系的数据;

列表 (list): 一对 `[]` 中包含由 `,` 分割的一系列元素，一个列表中可以包含任意多个 ($>=0$) 任意类型的数据，每一个数据称为一个元素。

创建列表的两种常用方式：

① 最简单方法是将列表元素放在一对方括号 `[]` 内，各元素之间以逗号分隔，并用 `=` 将一个列表赋值给某个变量。如，`list1 = [1,3,5]`；

② 使用内置的 `list` 函数创建列表，如 `list2 = list('Python')`，`list2` 的内容就是列表 `['P', 'y', 't', 'h', 'o', 'n']`。

注意： Python 允许同一个列表中各元素的数据类型不相同，可以是整数、字符串等基本类型，也可以是列表、集合及其他类型的对象，如 `list3 = ['python', True, 99, ['good', 'boy']]`。

列表访问

可以通过索引访问列表 (长度为 n) 中的元素，索引分为正向索引 (范围为: $0 \sim n - 1$) 和反向索引 (范围为: $-1 \sim -n$)，如下图所示：

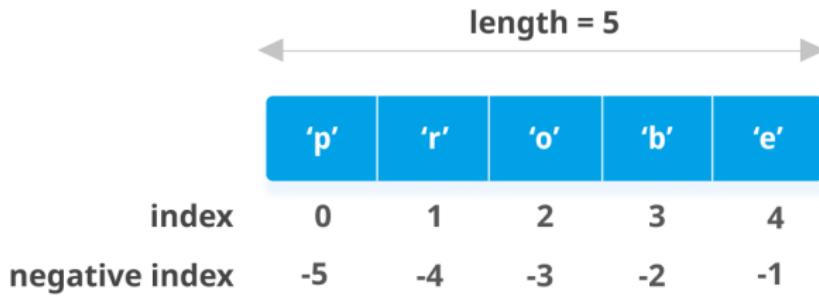


图 3: 列表的正向索引和反向索引

列表元素访问语法为 `list[index]`，称为下标变量，可用于读取 (读取列表中索引 $index$ 对应的元素值) 或写入 (更改列表中索引 $index$ 对应的元素值) 列表内容。

成员判断和切片

使用 `in` 和 `not in` 运算符可以判断一个元素是否在列表中。

列表切片：使用语法 `list[start:end]` 返回列表 `list` 的一个片段，其结果是 `list` 中索引 `start` 到 `end-1` 对应的元素所构成的一个新列表。

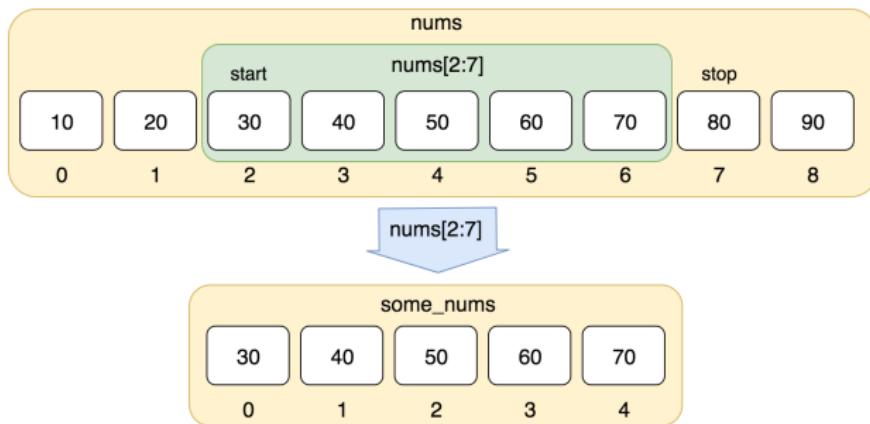


图 4: 列表的切片操作

26. 列表的基本操作

在切片操作中，起始下标 `start` 和结束下标 `end` 可以省略：① 如省略 `start`，则起始下标默认为 0，即从列表第一个元素开始截取；② 如省略 `end`，则结束下标默认为列表长度 n ，即截取到列表最后一个元素。下图中第三个参数 `-2` 表示切片步长，步长为正，要求从左往右切列表，步长为负，要求从右往左切列表。`list[::-1]` 可以实现列表快速翻转。

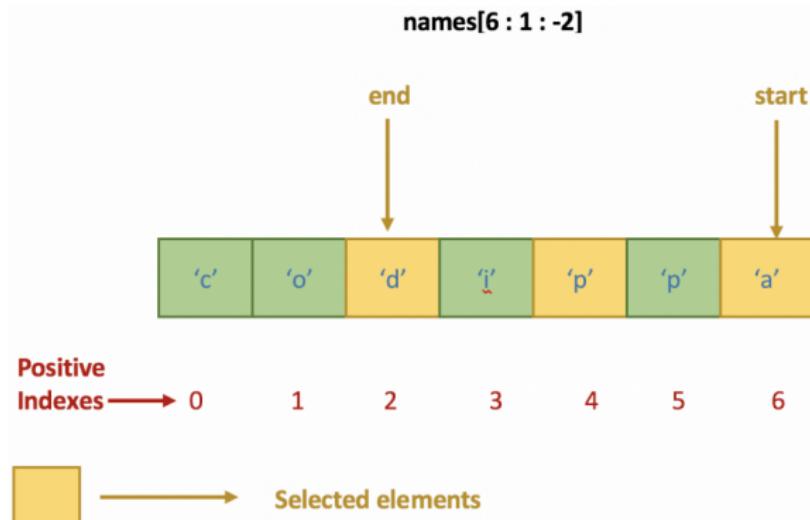


图 5: 带步长的列表的切片操作

修改列表元素值

可以通过**列表元素访问**和**切片操作**修改列表单个元素的值和同时修改多个元素的值，如下：

```
x = [1,2,3,4,5,6]
```

```
# 通过索引访问列表元素，修改单个元素值  
x[2] = 100 # [1,2,100,4,5,6]
```

```
# 通过列表切片，同时修改多个元素值  
x[-1:-3] = [200, 300] # [1, 2, 100, 4, 300, 200]
```

因为列表切片的结果是一个**列表**，所以上面切片赋值操作中等号右侧的内容一定是一个列表。

通过切片扩展、删减和复制列表

```
x = [1, 'a', 'b', 'c', 5, 6]

# 扩展列表
x[1:1] = [7, 8, 9] # [1, 7, 8, 9, 'a', 'b', 'c', 5, 6]

# 删减列表
x[1:4] = [] # [1, 'a', 'b', 'c', 5, 6]

# 复制列表
y = x[:] # [1, 'a', 'b', 'c', 5, 6]

# 引用赋值-(不推荐使用)
z = x # [1, 'a', 'b', 'c', 5, 6]

# 判断三个变量的值以及指向的内存空间是否相同
x == y == z # True
x is y # False
x is z # True
```

列表的拼接和复制

Python 中，使用 `+` 号拼接两个列表，并返回一个新列表，使用 `*` 复制一个列表若干次，返回一个新列表，如下：

```
x = [1, 2, 3]
```

```
y = x*3 # [1, 2, 3, 1, 2, 3, 1, 2, 3]
```

```
x[0] = 0 # 复制返回的是新列表，故y中的元素保持不变
```

```
z = x*2 + y # [0, 2, 3, 0, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]
```

```
# 复制和拼接都是返回新列表，故z中的元素保持不变
```

```
x[0] = 111
```

```
y[0] = 222
```

```
# [0, 2, 3, 0, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]
```

```
print(z)
```

列表的比较

使用关系运算符 `<, >, ==, <=, >=, !=` 对比较两个列表，规则为：

- ① 比较两个列表的第 0 个元素，如果两个元素相同，则继续比较下面两个元素；
- ② 如果两个元素不同，则返回两个元素的比较结果；
- ③ 一直重复这个过程直到有不同的元素或比较完所有的元素为止（长的列表大于短的列表，如下面的 `x < z`）。

```

x = [1, 2, 3, 4, 5]
y = [1, 2, 3, 100]
z = [1, 2 ,3 ,4 ,5, 6]
x < y # True
x < z # True
# 比较两个字符的Unicode码值, ord('阿') > ord('a'), 因此结果为True
xstr = ['阿', 'z']
ystr = ['a', '里']
xstr > ystr # True

```

列表推导式

列表推导式是一个生成新列表的简洁方法。一个列表推导式由方括号括起来，方括号内包含一个表达式和至少一个 `for` 循环语句，之后可以接 0 到多个 `for` 或 `if` 子句。列表推导式可以产生一个由表达式求值结果作为元素的新列表。

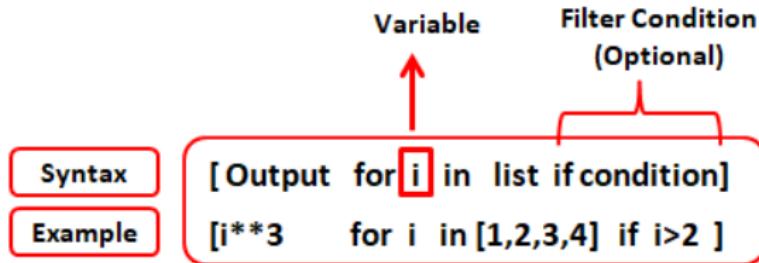


图 6: 列表推导式

- `all(iterable)` , 全部元素为 True, 返回 True;
- `any(iterable)` , 有一个元素为 True, 返回 True;
- `list(s)` , 将可迭代对象 s 变为列表;
- `len(s)` , 计算列表的长度;
- `max(iterable)` , 返回列表中最大的元素;
- `min(iterable)` , 返回列表中最小的元素;
- `sorted(iterable[, cmp[, key[, reverse]]])` , 对列表进行排序;
- `sum(iterable[, start])` , 对列表进行求和;
- `reversed(iterable)` , 翻转列表。

上述内置函数不仅可以用于操作列表, 还可用于操作其他可迭代对象, 使用 `help` 函数可查看其详细用法, 如 `help(all)`。

- `list.append(x)` , 在列表末尾增加一个元素 x;
- `list.extend(L)` , 在列表末尾再拼接一个列表 L;
- `list.insert(i, x)` , 在索引为 i 的位置插入一个元素 x;
- `list.remove(x)` , 从列表中移除元素 x;
- `list.pop(i)` , 删除索引 i 对应的元素;
- `list.index(x)` , 返回元素 x 对应的索引值;
- `list.count(x)` , 计数元素 x 在列表中出现的次数;
- `list.sort(key=None, reverse=False)` , 对列表进行排序;
- `list.reverse()` , 将列表翻转。

上述常用的列表方法只能用于操作列表，使用 `help` 函数可查看其详细用法，如 `help(list.append)`，`help(list)` 查看列表的全部方法。

- 基本用法：操作列表的内置函数的用法为 `function_name(list)`，
列表方法的用法为 `list.method_name(parameters)`；
- 结果：操作列表的内置函数不改变列表的内容，如 `reversed(list)` 其
返回结果是一个新的列表，而列表 `list` 不发生变化，列表方法会直
接改变表列表的内容，如 `list.reverse()` 执行后，列表 `list` 中的
元素将会翻转，并不会返回一个新列表。

注意：当元素 `x` 在列表 `list` 中多次出现时，`list.remove(x)` 只能
移除第一个 `x`(索引最小的那个 `x`)，`list.index(x)` 只能返回第一个 `x`
的索引值 (`x` 对应的最小索引值)。

30. 多维列表

多维列表：列表中嵌套一个或多个列表，比较常用的是二维列表（矩阵）和三维列表（张量）。

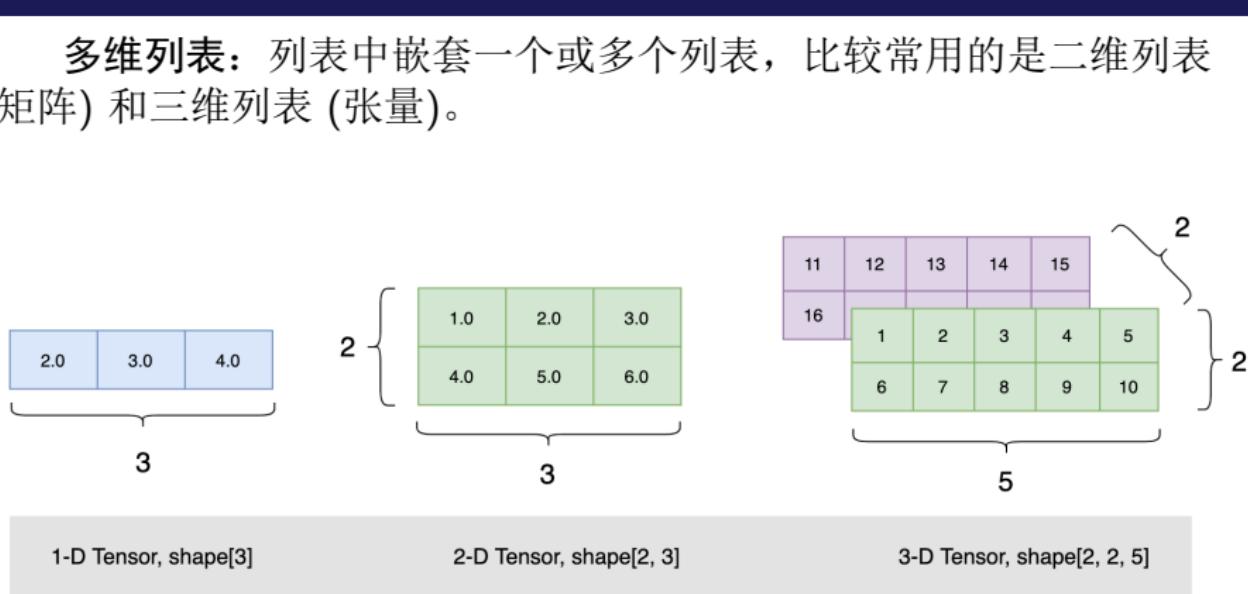


图 7：多维数值列表（张量）

访问多维列表中的元素：① 使用多重 `for` 循环遍历每个元素，② 使用多级索引访问某个元素，`tensor3[1][0][2]` 可访问上图最右侧三维列表中的元素 13，`tensor3[0][1][-2]` 返回 9。

- 多维列表本质也是一个列表，可通过 `type` 函数可查看其数据类型，其包含的元素也是列表；
- 前述的操作列表的内置函数和列表方法也适用于多维列表；
- 多维列表可以表示数学上的张量，也可以表示其他数据，如二维表（数据库表，简单 Excel 表格），其每一个维度值是灵活的，如下例：

```
# 下面的多维列表表示了下图中矩阵的左上角  
left_top = [[1, 2, 3], [4, 5], [7]]
```

1	2	3
4	5	6
7	8	9

在 Python 中进行数值运算，通常使用科学计算包 `NumPy` 中的 `array` 数据类型来表示矩阵、张量，比用多维数值列表更加计算高效。

元组 (**tuple**): 一对 () 中包含由 , 分割的一系列元素，一个元组中可以包含任意多个 (>=0) 任意类型的数据，每一个数据称为一个元素，元组中的元素不可变。

创建元组的两种常用方式:

① 最简单方法是将元组元素放在一对圆括号 () 内，各元素之间以逗号分隔，并用 = 将一个元组赋值给某个变量。如， `tuple1 = (1,3,5);`

② 使用内置的 **tuple** 函数创建元组，

如 `tuple2 = tuple('Python')`， `tuple2` 的内容就是元组 ('P', 'y', 't', 'h', 'o', 'n')。

注意: Python 允许同一个元组中各元素的数据类型不相同，可以是整数、字符串等基本数据类型，也可以是列表、集合及其他类型的数据，如 `tuple3 = ('python', True, 99, ['good', 'boy'])`。

正确理解元组中的数据元素不可变

① 元组一旦创建，其长度不可变，不能删减、增加元素，不能重新赋值元素；

② **不能重新赋值元素：**元组中的元素是不可变数据类型时，该元素不可变（重新赋值），如 1-Introduction.pdf 中介绍过的 basic 数据类型（整数、浮点数、字符串、布尔值、空值）；如果元素是可变数据类型，则可以改变该元素的值，但是不可以改变该元素的数据类型（重新赋值），具体看下面示例：

```
tuple4 = (1, 2, [1, 2, 3])
```

```
tuple4[2].append(4) # (1, 2, [1, 2, 3, 4])
```

```
# Error: 'tuple' object does not support item assignment
tuple4[2] = True
```

```
# Error: 'tuple' object does not support item assignment
tuple4[0] = 'good'
```

元组相关操作

可以使用 `tuple` 函数将列表、字符串等转换为元组；元组也是序列，一些用于列表的基本操作也可以用在元组上，如索引访问、成员判断、切片等。

```
# 使用tuple函数将列表转换为一个元组
list_1 = [1, 'Lisa', True, 6.6]
tuple_2 = tuple(list_1)

# 使用tuple函数将字符串转换为一个元组
string_1 = '阿里巴巴商学院'
tuple_3 = tuple(string_1)

# 判断元素是否在元组中
'巴' in tuple_3
# 使用索引访问元组中的元素
tuple_3[0]
# 对元组进行切片
tuple_3[1:3]
```

元组封装：将多个值自动封装到一个元组中；

序列拆封：将一个封装起来的序列自动拆分为若干个基本数据。

为多个变量同时赋值：结合了元组封装和序列拆封操作。

a, b, c, d = 4, 'Tony', True, 6.6 等价

于 my_tuple = 4, 'Tony', True, 6.6 和 a, b, c, d = my_tuple

```
# 元组封装
```

```
tuple_4 = 4, 'Tony', True, 6.6 # (4, 'Tony', True, 6.6)
```

```
# 元组拆封
```

```
a, b, c, d = tuple_4
```

```
# 列表拆封
```

```
list_2 = [8, 'Dan', False, 8.8]
```

```
a, b, c, d = list_2
```

相同点：均为序列数据类型，除元素修改操作外，其他操作，如索引访问、计数、切片等，两者用法相同；

不同点：元组属于不可变序列，一旦创建，便不允许进行元素的增加、删除和重新赋值，因此，元组没有 `append`、`extend`、`insert` 方法；列表是可变序列，可以对其中元素进行增、删、改操作；

应用场景不同：元组的访问和处理速度比列表更快。因此，如果所定义的序列内容不会进行修改，最好使用元组而不是列表。另外，使用元组也可以使元素在实现上无法被修改，从而使代码更加安全。

标准的序列操作同样适用于字符串：

- 索引，切片
- 乘法，加法
- 成员资格判断
- 求长度，最大值，最小值

```
str1 = '喜欢看你紧紧皱眉 叫我胆小鬼'  
str1[-3:] # 胆小鬼  
'皱眉' in str1 # True  
len(str1), max(str1), min(str1) # (14, '鬼', ' ')  
'鲁班' + '大师' # 鲁班大师  
'娜可' + '露'*2 # 娜可露露
```

字符串不可变，因此无法进行索引赋值或切片赋值。

35. 字符串常用方法

可以使用内置函数 `help` 查看字符串的全部方法，命令为 `help(str)`。

```
# find函数返回子串在字符串中第一次出现的索引位置, 没找到则返回-1
str9 = "My heart will go on and on"
str9[14:20] # 'go on'
str9.find('on', 14, 20) # 17, 即使指定查找范围,
→ 返回的索引值仍然是在整个字符串中的索引值
# index函数返回子串在字符串中第一次出现的索引位置,
→ 没找到则代码报错 substring not found
str9.index('on', 14, 20) # 17 即使指定查找范围,
→ 返回的索引值仍然是在整个字符串中的索引值
str9.index('on', 0, 8) # 没找到, 报错
# startswith, endswith分别用于判断字符串是否以特定字符为开头和结尾
str9.startswith('My') # True
str9.endswith('and',0,-3) # True
# count函数计算子串在字符串中出现的次数
"My heart will go on and on".count('n',-3) # 1, start和end可以省略
```

`startswith`, `endswith`, `find`, `index` 均可以用索引值指定查找的范围。`find` 找不到返回-1, `index` 找不到则报错。

常见的 4 个空白字符有: \t , \r , \n , 空格。

```
# center函数令字符串居中，并指定宽度和填充字符
"回到最初我们来的地方".center(20, '~') #
→ ~~~~~回到最初我们来的地方~~~~~
```

split函数将字符串分割为列表，默认分隔符为空白字符

join函数将字符串列表组合为字符串，默认连接符为空字符，
→ 空字符就是两个引号之间啥也没有，不是空格，也不是空白字符

```
str9.split(' ') # ['My', 'heart', 'will', 'go', 'on', 'and',
→ 'on']
'~'.join(str9.split(' ')) # My~heart~will~go~on~and~on
```

strip方法去除字符串开头和结尾的指定字符，默认去除空白字符

```
"let's goooooooo".strip('o') # let's g
"\tAlibaba\r\n".strip() # Alibaba
```

其他常用的字符串方法如下：

```
# lower和upper分别将字符串中的全部字符变为小写和大写
'My heart'.lower() # 'my heart'
'My heart'.upper() # 'MY HEART'
# replace将字符串中的特定字符替换为指定字符
my_str = 'so good'
my_str.replace('o', '~') # s~ g~~d, my_str内容不变
'goooooood'.replace('ooo', '') # good
# translate方法根据定义的转换表转换字符串的字符
# 1. 定义转换表
intab = "aeiou"
outtab = "12345"
trantab = str.maketrans(intab, outtab)
# 2. 使用转换表进行批量替换
str10 = "this is string example....wow!!!"
# th3s 3s str3ng 2x1mpl2....w4w!!!
print(str10.translate(trantab))
```

使用字符串的下列内置方法判断字符串是否满足特定的条件：
isalnum、isalpha、isdecimal、isdigit、isidentifier、islower、isnumeric、
isprintable、isspace、istitle、isupper， 请自行敲代码学习这些方法。

```
string = "Python3"
print(string.isalnum())  # True

string = "Python"
print(string.isalpha())  # True

string = "12345"
print(string.isdigit())  # True

string = "python"
print(string.islower())  # True

string = "Python Programming"
print(string.istitle())  # True
```

string 模块包含了许多常用的字符集：

```
import string

# abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
print(string.ascii_letters)
# abcdefghijklmnopqrstuvwxyz
print(string.ascii_lowercase)
# ABCDEFGHIJKLMNOPQRSTUVWXYZ
print(string.ascii_uppercase)
# 0123456789
print(string.digits)
# !#$%&'()*+,-./:;<=>?@[\\]^_`{|}
print(string.punctuation)
```

Python 的内置函数 `str` 将对象转化为人类可读的字符串表示，`repr` 将对象转化为 Python 解释器读取的字符串表示，两个函数返回的结果都是字符串 (string) 类型。

当字符串作为两者的参数时，`repr` 返回的结果会多一层引号。

```
s = '1*2*3*4'  
str(s) # '1*2*3*4'  
repr(s) # "'1*2*3*4'"
```

`eval` 函数接受一个字符串作为参数，其作用是把字符串中的内容作为 Python 语句运行，并返回运行结果 (即，剥离引号 + 运行命令)。

```
eval(str(s)) # 24  
eval(repr(s)) # '1*2*3*4'
```

37. 转义字符

要在字符串中插入“特殊字符”，如，引号、右斜线、换行符等，需使用转义字符，Python 中常用的转义字符如下图所示：

转义字符	说明
\n	换行符，将光标位置移到下一行开头。
\r	回车符，将光标位置移到本行开头。
\t	水平制表符，也即 Tab 键，一般相当于四个空格。
\a	蜂鸣器响铃。注意不是喇叭发声，现在的计算机很多都不带蜂鸣器了，所以响铃不一定有效。
\b	退格 (Backspace)，将光标位置移到前一列。
\\"	反斜线
\'	单引号
\"	双引号
\	在字符串行尾的续行符，即一行未完，转到下一行继续写。

在字符串的引号前面加上 `r` 可消除字符串中的转义 (如存在)。

```
str1 = "\\"是一个双引号"
print(str1) # "是一个双引号
print('Hello\nProgrammers')
print(r'Hello\nProgrammers')
print(r"\\"是一个双引号") # \"是一个双引号
```

什么是字符串格式化

字符串格式化本质上就是一个字符串模板，用于高效（模板的功能性）、美观（模板的装饰性）地生成可打印的字符串。

高效：在字符串格式化语句中，公共的、不变的部分只书写一次，可变的部分使用字符串格式化方法进行动态填充，例如下面使用字符串格式化打印青蛙跳水儿歌，可变部分是与青蛙数量相关的数量值，其他部分不变。

```
# num取值从1-9
for num in range(1,10):
    str_tmp = '{}只青蛙{}张嘴，{}只眼睛{}条腿，'.format(num, num,
    ↳ 2*num, 4*num) + '扑通~'*num + '{}声跳下水'.format(num)
    print(str_tmp)
```

输出结果见下页。

什么是字符串格式化 (续)

1只青蛙1张嘴，2只眼睛4条腿，扑通~1声跳下水

2只青蛙2张嘴，4只眼睛8条腿，扑通~扑通~2声跳下水

3只青蛙3张嘴，6只眼睛12条腿，扑通~扑通~扑通~3声跳下水

4只青蛙4张嘴，8只眼睛16条腿，扑通~扑通~扑通~扑通~4声跳下水

5只青蛙5张嘴，10只眼睛20条腿，扑通~扑通~扑通~扑通~扑通~5声跳下水

6只青蛙6张嘴，12只眼睛24条腿，扑通~扑通~扑通~扑通~扑通~扑通~6声跳下水

7只青蛙7张嘴，14只眼睛28条腿，扑通~扑通~扑通~扑通~扑通~扑通~7声跳下水

8只青蛙8张嘴，16只眼睛32条腿，扑通~扑通~扑通~扑通~扑通~扑通~扑通~8声跳下水

9只青蛙9张嘴，18只眼睛36条腿，扑通~扑通~扑通~扑通~扑通~扑通~扑通~扑通~9声跳下水

图 8: 使用字符串格式化高效打印青蛙跳水儿歌

什么是字符串格式化 (续)

美观: 字符串格式化使用一系列格式说明符指定所填充内容的显示格式, 例如, 下面的例子用字符串格式化整齐美观地(模板的装饰性)打印出商品价格表。

```
width = int(input('Please enter width: '))
price_width = 10
item_width = width - price_width
header_fmt = '{{:{}}}{{{:>{}}}}'.format(item_width, price_width)
fmt = '{{:{}}}{{{:>{}}.2f}}'.format(item_width, price_width)
print('=' * width)
print(header_fmt.format('Item', 'Price'))
print('-' * width)
print(fmt.format('Apples', 0.4))
print(fmt.format('Pears', 0.5))
print(fmt.format('Cantaloupes', 1.92))
print(fmt.format('Dried Apricots (16 oz.)', 8))
print(fmt.format('Prunes (4 lbs.)', 12))
print('=' * width) # 50
```

什么是字符串格式化 (续)

```
Please enter width: 50
=====
Item                Price
-----
Apples              0.40
Pears               0.50
Cantaloupes         1.92
Dried Apricots (16 oz.) 8.00
Prunes (4 lbs.)    12.00
=====
```

图 9: 使用字符串格式化整齐美观地打印出价格表

总结: 字符串格式化的语法包括两部分: ① 占位符 (% 或 {}), 指定在哪里填充内容, 不可省略; ② 格式说明符, 指定填充内容的显示格式, 可以省略。如果只使用字符串格式化的功能性, 可以不写格式符, 只使用占位符即可, 如果也使用字符串格式化的装饰性, 那就需要根据需求指定格式说明符。

字符串格式化：用元组内的元素按照预设格式替换字符串中的内容。

两种方式：使用 %；使用 `format` 方法。

```
num = 3
str2 = '%s只青蛙%s张嘴, %s只眼睛%s条腿, ' %(num, num, 2*num, 4*num)
→ + '扑通~'*num + '%s声跳下水' %(num)
print(str2)

num = 5
str3 = '{}只青蛙{}张嘴, {}只眼睛{}条腿, '.format(num, num, 2*num,
→ 4*num) + '扑通~'*num + '{}声跳下水'.format(num)
print(str3)
```

% 为占位符，标记了需要插入值的位置。格式符号 s 表示插入的值会被格式化为字符串，如果不是字符串会自动转换为字符串（如，上例中插入的 num 为整数）。

`format` 方法可以按照从左到右的顺序依次将元组中的值插入到占位符 {} 所在的位置。

39. 字符串格式化功能

字符串格式化方法 `format`，增强了字符串格式化的功能。

1. 按照索引进行替换，顺序（推荐）

```
num = 5
str3 = '{}只青蛙{}张嘴, {}只眼睛{}条腿, '.format(num, num, 2*num,
→ 4*num)
print(str3)
```

2. 按照索引进行替换，乱序（不推荐）

```
singer_template = "歌手{3}, {0}年出生于{1}, 代表作有《{2}》等。"
print(singer_template.format(1978, '新加坡', '胆小鬼', '孙燕姿'))
```

3. 按照字段名进行替换（推荐）

```
singer_template = "歌手{name}, {birth_year}年出生于{nation},
→ 代表作有《{masterpiece}》等。"
print(singer_template.format(birth_year = 1963, nation =
```

```
→ '中国天津市', masterpiece = '好汉歌', name = '刘欢'))
```

4. 字段名和索引可以混用，元组中位置参数（索引）在前，

→ 关键字参数（字段名）在后（不推荐）

```
singer_template = "歌手{name}, {0}年出生于{nation}, 代表作有《{1}》
→ 等。"
print(singer_template.format(1991, '光年之外', nation =
```

```
→ '中国上海市', name = '邓紫棋'))
```

`format` 方法中的占位符 `{}` 的完整形式为 `{字段名或索引: 字段格式}`，冒号前后的内容均可省略，若都省略，则按顺序将元组中的内容以字符串格式进行显示。

```
# /表示填充字符，如果不指定默认就是空格填充;
# ^表示居中显示
# +表示在正数前面加正号，负数前面加负号
# 10表示宽度为10
# .2f表示保留两位小数的浮点数
str4 = "{:/^+10.2f}"
str4.format(3.1415926) # //+3.14///
```

40. 字符串格式化的装饰性

标准的字符串格式说明符如下，`[]` 表示其中的参数是可选的：

```
[[fill]align][sign][#][0][width][grouping_option][.precision][type]
```



图 10：类比游戏中的装备格理解标准的字符串格式说明符

- ① 游戏中装备格子有限，每个装备格子不一定非得买装备；格式说明符类型也有限（数数上面有几对中括号），每类格式说明符不一定被指定；
- ② 游戏中有出装顺序；标准格式说明符也有顺序呀（上面从左到右就是顺序）；
- ③ 游戏中可以一件装备也不出（没输出、不抗揍、跑得慢）；格式说明符也可以一个也不指定（没什么装饰性了）；
- ④ 游戏中不带惩戒不能买打野刀；格式说明符里不指定 `align`（对齐方式）就不能指定 `fill`（填充字符）。

① [fill]align 参数

align (对齐方式)	作用
<	左对齐 (字符串默认对齐方式)
>	右对齐 (数值默认对齐方式)
=	填充时强制在正负号与数字之间进行填充，只支持对数字的填充
^	居中

图 11: align 参数

除非定义了字段宽度，否则字段宽度将始终与填充它的数据大小相同，因此，在未指定字段宽度的情况下，对齐选项没有意义。

fill 指定填充字符。填充字符 (如果存在) 必须后跟对齐标志。

② **sign** 参数，仅对数字类型有效：

sign	作用
+	强制对数字使用正负号
-	仅对负数使用前导负号(默认)
空格	对正数使用一个' '作前导，负数仍以'-'为前导

图 12: sign 参数

如果存在 # 字符，则整数使用“替代形式”进行格式化。这意味着二进制、八进制和十六进制输出将分别以“0b”、“0o”和“0x”为前缀。

③ **width** 参数，定义字段宽度。如果未指定，则字段宽度将由填充的内容的长度决定。如果 **width** 前面有一个零 0 字符，则对数字启用零填充。

- ④ **grouping_option**，指定数字的整数部分使用的千分位分隔。

描述符	作用
,	使用,作为千位分隔符
—	使用_作为千位分隔符

图 13: grouping_option 参数

⑤ **.precision**，精度参数，是一个十进制数，若填充的内容为浮点数，表示保留的小数位数；若填充内容为字符串，表示填充的字符个数；若填充的内容为整数，该参数不可用。

⑥ **type** 决定了数据应该如何呈现，默认值为 **s**，以字符串格式显示。

对于 `type` 参数，若填充的内容为整数，还可以使用如下选项：

- `b`，以二进制显示；
- `c`，显示数字对应的 `unicode` 字符；
- `d`，以十进制显示；
- `o`，以八进制显示；
- `x` 或 `X`，以十六进制显示。

对于 `type` 参数，若填充的内容为浮点数，还可以使用如下选项：

- `e` 或 `E`，用科学计数法表示，使用 `E` 或 `e` 表示指数；
- `f` 或 `F`，用普通浮点数表示，对于 `nan(not a number)` 和 `inf(无穷大)` 用大 (小) 写表示；
- `g` 或 `G`，自动在科学计数法和普通浮点数表示中做出选择，指数 `E(e)` 大小写取决于 `G(g)`；
- `%`，以百分数显示，即，对原数值乘以 100 后，加上 `%` 作后缀。

字符串格式化其他参考资料：[资料 1](#), [资料 2](#), [资料 3](#),

40. 字符串格式化的装饰性

```
str5 = "{:^10.5s}的成绩是{:<+10.2f}，班级排名第{:d}"  
str5.format("张三", 91.5, 5) # 张三 的成绩是+91.50 ,  
→ 班级排名第5  
str6 = "{num:+e}的二进制形式为{num:+#b}，八进制为{num:+#o},  
→ 十六进制为{num:+#X}"  
str6.format(num=456) # +4.560000e+02的二进制形式为+0b111001000,  
→ 八进制为+0o710, 十六进制为+0X1C8  
str7 = "{:5.3s}的年收入为{:5d}元，月平均{:5.2f}元"  
str7.format('zjzhang', 60023, 60023/12.0) # zjz  
→ 的年收入为60023元，月平均5001.92元  
str8 = "{:5.3s}的年收入为{:4d}元，月平均{:5.2f}元"  
str8.format('zjzhang', 60023, 60023/12.0) # zjz  
→ 的年收入为60023元，月平均5001.92元  
str8.format('Tomy', 67, 67/12.0) # Tom 的年收入为 67元，月平均  
→ 5.58元
```

- ① 字符串的精度是对字符串的直接切片 (看上例中的 str7);
- ② 浮点数精度存在四舍五入，首先保证小数部分，当长度超过宽度时，最初设置的宽度将不起作用 (看上例中的 str7); 整数不允许使用精度;
- ③ 默认对齐方式，字符串左对齐，数值右对齐 (看上例中的 str8);
- ④ 内部自转义： {{ 表示 {， }} 表示 }， %% 表示 %

```
# {}你看，我左边是什么
'{{}}{}'.format(text = '你看，我左边是什么')
# %是百分号
'%%%s'%(是百分号)
```

40. 字符串格式化的装饰性

f-string: 在形式上以 *f* 或 *F* 修饰符引领的字符串 (*f'xxx'* 或 *F'xxx'*)，以大括号 {} 标明被替换的字段，可在 {} 内调用变量、函数和表达式求值。

```
import math
radius = 3
# 有一个圆，半径为3.00，面积为28.27
f"有一个圆，半径为{radius:.2f}，面积为{math.pi*(radius**2):.2f}"  
  
x = 80
# 角度值为80.00，弧度值为1.39626，其正弦值为0.98
F"角度值为{x:.2f}，弧度值为{math.radians(x):.5f}，  
→ 其正弦值为{math.sin(math.radians(x)):.2f}"  
  
# 再来看一个结合字符串格式化的例子
num_str = '1 2 \r 3\t4\n9'
num_list = num_str.split() # ['1', '2', '3', '4', '9']
'*'.join(num_list) # 1*2*3*4*9
print(f"{'*'.join(num_list)}的计算结果为: {1*2*3*4*9}") #
→ 1*2*3*4*9的计算结果为: 216
```

集合：一对 {} 中包含由 , 分割的一系列元素。

- 集合可以包含任意多个、无序的非重复元素；
- 集合中的元素可以是多种基本数据类型，如字符串，整数，布尔值，None；
- 集合中的元素不可以是可变数据类型，如列表，字典；
- 可以使用 {} 定义集合，也可以使用 set 函数将字符串、元组、列表等序列转换为集合。

```
# 定义一个集合
set1 = {1, 2, (1,2), True}
# 使用set函数将字符串转换为字符集合
chr_set = set('zjzhang') # {'a', 'g', 'h', 'j', 'n', 'z'}
```

集合中的元素无序，故不能使用索引访问其中的元素，但可以使用 `for` 循环遍历其中的元素，亦可以使用成员判断操作符 `in` 判断某个元素是否在集合中。

- 内置函数 `len` 可统计集合中的元素个数；
- `max` 可返回集合中的最大元素；
- `sum` 可求数值集合中所有元素的和；

```
chr_set = set('zjzhang') # {'a', 'g', 'h', 'j', 'n', 'z'}
for char in chr_set:
    print(char)

'o' not in chr_set # True
```

集合常用的方法有：

- **add**，添加一个元素到集合中；
- **remove**，移除集合中的一个指定元素，不存在该元素时报错；
- **discard**，移除集合中的一个指定元素，不存在该元素时不报错；
- **pop**，随机移除集合中一个元素；
- **clear**，清空集合中的全部元素；

这些方法的运行结果会改变集合的内容。

```
chr_set = set('zjzhang') # {'a', 'g', 'h', 'j', 'n', 'z'}
chr_set.add('c') # {'a', 'c', 'g', 'h', 'j', 'n', 'z'}
chr_set.remove('a') # {'c', 'g', 'h', 'j', 'n', 'z'}
chr_set.remove('q') # 报错, KeyError
chr_set.discard('a') # {'c', 'g', 'h', 'j', 'n', 'z'}
chr_set.discard('z') # {'c', 'g', 'h', 'j', 'n'}
chr_set.pop() # 随机删除一个元素并返回该元素
chr_set.clear() # set()
```

Python 提供了求交集、并集、差集和对称差集等集合运算：

- 使用 `s1.intersection(s2)` 或者 `s1&s2` 可以计算两个集合的交集；
- 使用 `s1.union(s2)` 或者 `s1|s2` 可以计算两个集合的并集；
- 使用 `s1.difference(s2)` 或者 `s1-s2` 可以计算两个集合的差集；
- 使用 `s1.symmetric_difference(s2)` 或者 `s1^s2` 可以计算两个集合的对称差集。

字典 (dict): 一对 {} 中包含由 , 分割的一系列无序键值对 (key-value pair)，一个字典中可以包含任意多个键值对，字典使用关键字 (key) 来访问、更新、删除值 (value)。

```
# 定义一个字典
dict1 = {115:'张三', 116:'李四', 119:'王五'}
# 通过key访问字典中的值
dict1[115] # 张三
# 通过key更新字典中的值
dict1[115] = '张七' # {115: '张七', 116: '李四', 119: '王五'}
# 通过key删除字典中的值
dict1.pop(115) # {116: '李四', 119: '王五'}
# 字典中的值不一定是唯一的
dict1[120] = '王五' # {116: '李四', 119: '王五', 120: '王五'}
# 字典中的关键字是唯一的
dict1[120] = '王四' # {116: '李四', 119: '王五', 120: '王四'}
```

注意：字典中的关键字 (key) 是唯一的 (unique)，值不一定是唯一的，即一个关键字只能对应一个值，但是同一个值可以对应多个关键字。

类似于之前的列表、元组、集合，字典也可以使用 Python 内置函数 `dict` 创建，其接收的参数形式如下：

`[(key1, value1), (key2, value2), ...]`。

定义字典时，重复的键只保留其中一个，并且，键只能是不可变对象，如，之前学习过的基本数据类型（数值、字符串等）。列表、集合是可变对象，不能作为字典的键，字典的值（value）可以是任意的数据类型。

```
# 使用Dict函数定义一个字典
dict2 = dict([(116, '李四'), (119, '王五'), (120, '赵六')])
# 使用zip函数将两个列表转化为dict函数接收的参数形式
dict(zip([116, 119, 120], ['李四', '王五', '赵六'])) # {116:
→ '李四', 119: '王五', 120: '赵六'}
# 重复的键只会保留一个
dict3 = {115:'张三', 115:'李四', 115:'王五'} # {115: '王五'}
# 字典的键只能是不可变对象
dict4 = {[1,2]: [1,2]} # Error
# 字典的值可以是任意类型
dict5 = {'list1':[1,2,3]}
```

字典的增、删、改、查操作通过关键字 (key) 进行，类似于通过索引 (index) 操作列表。

```
# 创建一个空字典
happy_dict = {}
# 新增键值对
happy_dict[101] = '沈腾'
happy_dict.update({102: '马丽'})
happy_dict.update(dict(zip([103,104],['艾伦','常远'])))
happy_dict[105] = '宋小宝'
# 修改值
happy_dict[105] = '王宁'
# 通过键查找其对应的值
happy_dict[102]
happy_dict.get(103)
# 删除键值对
happy_dict.pop(105)
del happy_dict[104]
```

字典存储数据实例

```
# 创建一个字典 company_info, 存储一家公司信息
company_info = {'name': 'TechCorp', 'city':
→  'Shanghai', 'employees': 500}

# 增加一个键值对, 增加公司的年收入信息 -2000万
company_info['annual_revenue'] = 20000000

# 删除一个键值对, 删除公司的员工人数信息
del company_info['employees']

# 修改某个键对应的值, 修改公司地址为背景
company_info['city'] = 'Beijing'

# 根据键查看值的信息, 查看公司名称
company_info['name']

# 新增一个键值对, 值的数据类型为列表
company_info['products_services'] = ['软件开发', '数据分析',
→  '云服务']
```

字典存储数据实例

```
# 新增一个键值对，值的数据类型为字典
company_info['financial_status'] = {'资产': 300000000, '负债':
→ 100000000}

# 根据键查看值的信息，查看公司名称
# 如果键 'founder' 存在，则返回对应的值，如果不存在，返回 '未知'
company_info.get('founder', '未知')

# 根据键查看值的信息，查看公司名称
# 如果键 'founder' 存在，则返回对应的值，如果不存在，返回 'Tom',
→ 并新增键值对 founder:Tom
company_info.setdefault('founder', 'Tom')

# 更新字典信息，使用一个已有字典更新
company_info.update({'profit': 10000000, 'market_share': '20%'})

# 更新字典信息，使用一个二元组列表更新
company_info.update(zip(['brand_value', 'customer_satisfaction'],
→ [5000000, 4.5]))
```

字典推导式的基本形式同列表推导式，唯一不同的是表达式的格式，

key:value。

可以使用 **in** 或 **not in** 判断某个键 (key) 是否在字典中，可以使
用 **==** 判断两个字典中是否包含一样的键值对 (key-value pairs)。

```
actor_list = ['沈腾', '马丽', '艾伦', '常远', '王宁']
start_id = 101
happy_dict = {start_id+idx:actor for idx,actor in
    ↪ enumerate(actor_list)}
# {101: '沈腾', 102: '马丽', 103: '艾伦', 104: '常远', 105:
    ↪ '王宁'}

# 判断键是否在字典中
'沈腾' in happy_dict # False
102 in happy_dict # True

# 判断两个字典的内容是否一样
{101:'沈腾', 102:'马丽'} == {102:'马丽', 101:'沈腾'} # True,
    ↪ 字典是无序的
```

内置方法的语法为 变量名. 方法名

- `copy`，复制一份字典，不受原字典变化影响；
- `get`，根据键获取对应的值，键不存在，返回默认值；
- `setdefault`，根据键获取对应的值，键不存在，返回默认值，更新字典；
- `keys`，返回字典中的全部键；
- `values`，返回字典中的全部值；
- `items`，返回字典中的全部键值对；
- `pop`，删除字典中的指定键值对；
- `popitem`，以后进先出的顺序删除字典中的一个键值对；
- `update`，使用一个字典或二元组列表更新当前字典；
- `clear`，清空字典；

45. 字典的常用内置方法

```
happy_dict = {101: '沈腾', 102: '马丽', 103: '艾伦', 104: '常远',
→ 105: '王宁'}
new_happy_dict = happy_dict.copy()
happy_dict.get(101) # 沈腾
happy_dict[101] # 键不存在则Error, 存在则返回值
happy_dict.get(109, '不存在此键') # 键存在, 则返回对应的值,
→ 键不存在, 则返回第二个参数指定的值, 第二个参数默认值为None
happy_dict.setdefault(101) # 沈腾
happy_dict.setdefault(106, '魏翔') # 键存在, 则返回对应的值,
→ 键不存在, 则返回第二个参数指定的值, 第二个参数默认值为None,
→ 同时更新字典
# {101: '沈腾', 102: '马丽', 103: '艾伦', 104: '常远', 105:
→ '王宁', 106: '魏翔'}
happy_dict.keys() # dict_keys([101, 102, 103, 104, 105, 106])
happy_dict.values() # dict_values(['沈腾', '马丽', '艾伦',
→ '常远', '王宁', '魏翔'])
happy_dict.items() # dict_items([(101, '沈腾'), (102, '马丽'),
→ (103, '艾伦'), (104, '常远'), (105, '王宁'), (106, '魏翔')])
happy_dict.pop(106) # 返回106对应的值, 魏翔, key不存在, 则Error
happy_dict.popitem() # 返回被删除的键值对, (105, '王宁')
```

```
happy_dict.update({107: '杜晓宇', 108: '黄才伦'})  
happy_dict.update({108: '魏翔'}) # 更新108对应的值,  
→ 由'黄才伦'变为'魏翔'  
happy_dict.clear() # happy_dict变为一个空字典
```

```
# {}表示空字典  
d = {}  
type(d) # dict  
# 空集合是set()  
s = {1,2}  
type(s) # set  
s.clear()  
print(s) # set()  
type(s) # set
```

Python 中的赋值语句不复制对象，它们在目标 (变量) 和对象 (数据) 之间创建绑定。

对于可变 (如，列表) 或包含可变项的复合对象 (如，列表嵌套)，有时需要一个副本，以便可以更改一个副本而不更改原数据对象。

注意：浅复制和深复制之间的区别仅与复合对象有关，典型的复合对象有字典中包含列表，列表中包含列表，元组中包含列表作为元素。

```
# 创建一个列表嵌套对象
list1 = [['John', 21], ['Mary', 19]]
# 引用赋值
list2 = list1
# 深复制
import copy
list3 = copy.deepcopy(list1)
# 浅复制-切片操作
list4 = list2[:]
# 浅复制-列表推导式
list5 = [item for item in list1]
```

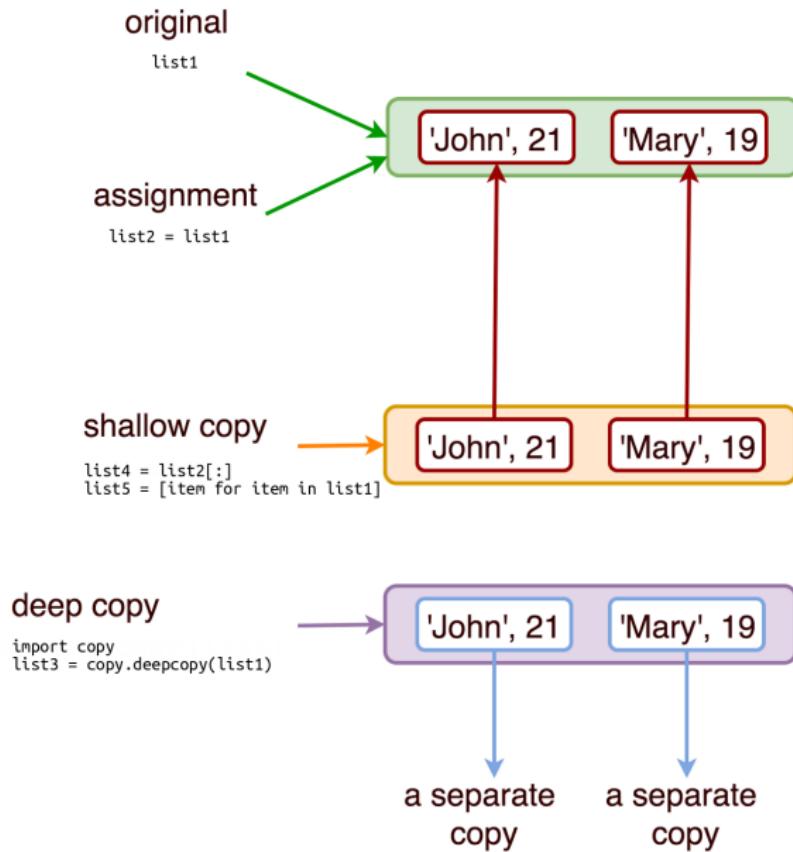


图 14: 赋值、深复制、浅复制示例

```
list1[0][0] = 'Tom' # [['Tom', 21], ['Mary', 19]]
print(list2) # [['Tom', 21], ['Mary', 19]]
print(list3) # [['John', 21], ['Mary', 19]]
print(list4) # [['Tom', 21], ['Mary', 19]]
print(list5) # [['Tom', 21], ['Mary', 19]]  
  
list2 is list1 # True
list3 is list1 # False
list4 is list1 # False
list5 is list1 # False  
  
list5[0] is list1[0] # True
list4[0] is list1[0] # True
list3[0] is list1[0] # False
list2[0] is list1[0] # True
```

对于包含可变项的复合对象，只有 `copy` 模块的 `deepcopy` 才能实现深复制，其他复制操作均为浅复制，如，切片、推导式等，字典的内置方法 `copy` 也是浅复制；

注意：包含可变项的复合对象是可变数据类型，也不能作为集合的元素，不能作为字典的键 (key)，如，`(1,2,[3,4])`；

如果一个字典中包含列表作为值，各种复制操作的结果与代码中的列表嵌套示例类似。请自行实验

对 `dict6 = {101:['Tom', 'Mary', 'Alan']}` 的各种复制操作并查看相应结果。

使用 Python 的一些内置函数辅助了解相关模块和函数的用法：

- 使用内置函数 `dir` 可以快速查询各种内置数据类型的方法有哪些；
- 使用特殊变量 `__doc__` 可以快速查看各种方法的使用说明；
- 使用内置函数 `help` 查看函数或模块用途的详细说明；
- 使用内置函数 `type` 查看对象的数据类型；

- **顺序结构：**按照书写顺序依次解释执行；
- **选择结构：**按照判断条件选择其中一个分支执行；
- **循环结构：**满足指定条件时重复执行某些操作。

```
# 顺序结构，先接收键盘输入，再打印输出
age = float(input("Please input your age: "))
print("Your age is {}".format(age))
```

```
# 选择结构，年龄小于18，禁止购买香烟
if age < 18:
    print("Sorry, we can not sell cigarettes to you!")
else:
    print("Please input your ID number.")
```

```
# 循环结构，如身份证号不是18位数字，则重新输入
while True:
    id_num = input("ID: ")
    if id_num.isdigit() and len(id_num) == 18:
        print('正在与公安系统联网核验....., 请稍后')
        break
```

单选结构

```
# 下面为伪代码，不可直接运行
if condition:
    statements1

other statements
```

当条件判断表达式 (condition) 返回结果为 `True` 时，执行语句块 1 (`statements1`)，否则 (`False`)，直接顺序往下运行与 `if` 对齐的后续代码，在上例中，如果 `condition` 计算结果返回 `False`，则直接执行后续代码 (`other statements`)。

注意：① 判断条件 (condition) 必须是一个能返回布尔值 (或等价于布尔值，如，`0`, `1`, `None`) 的表达式，如，比较运算和逻辑运算表达式；
② 语句块 1 (`statements1`) 可以包含任意多行代码。

```
# 互换两个变量的值，按照从小到大的顺序输出两个值
a = float(input("Please input the first integer: "))
b = float(input("Please input the second integer: "))
print('Before exchange:', a, b)
if a > b: # 条件
    a, b = b, a # 语句块
print('After exchange:', a, b) # 单选结构外的语句

# 寻找180+的男孩子
height = input("Please input your height(cm): ")
if float(height) < 180:
    print("Sorry, you are a good boy, but you know ...")
print('Nice to meet you ... Let\'s talk about the homework 4.')
```

双选结构

```
# 下面为伪代码，不可直接运行
if condition:
    statements1
else:
    statements2

other statements
```

当条件判断表达式 (condition) 返回结果为 `True` 时，执行语句块 1 (`statements1`)，否则 (`False`) 执行语句块 2 (`statements`)，最后再顺序执行后续的语句块 (`other statements`)。

注意：双选结构中 `statements1` 和 `statements2` 有且只有一个被运行。

```
# 根据身份证号判断性别，第15-17位数字组成的三位数如果是奇数则为男性
last_four = input("Please input the last four number of your ID:
→   ")
num = int(last_four[:-1])
if num % 2:
    print('Male')
else:
    print('Female')
```

上述条件 `num % 2` 返回 0 或者 1，等价于 `False` 或 `True`。

多选结构

```
# 下面为伪代码，不可直接运行
if condition1:
    statements1
elif condition2:
    statements2
...
elif conditionN:
    statementsN
...
else:
    statements0

other statements
```

满足条件 1 (condition1) 执行语句块 1 (statements1)，满足条件 2 (condition2) 执行语句块 2 (statements2)，……，所有条件都不满足，执行语句块 0 (statements0)，执行完选择结构中的某一个语句块后，立刻离开选择结构，执行后续代码 (other statements)。

```
# 根据分数输出相应评语
score = float(input("Please input your final score:"))

if 90 <= score <= 100:
    print('Great!')
elif 80 <= score < 90:
    print('Good!')
elif 70 <= score < 80:
    print('You could be better!')
elif 60 <= score < 70:
    print('Dangerous!')
elif 0<= score < 60:
    print("It's a pity!")
else:
    print('The score is invalid!')
```

注意：使用多选结构时，要根据实际问题，全面考虑可能的条件判断，如，上例中，百分制下，合法的分数应该在 0 到 100 之间。

选择结构嵌套: 条件满足时 (`True`) 所执行的语句块 (`statements`) 也是一个选择结构。

```
amount = float(input("输入驾驶员每100ml血液酒精的含量(mg): "))
if amount < 20:
    print("驾驶员不构成酒驾")
else:
    if amount < 80:
        print("驾驶员已构成酒驾")
    else:
        print("驾驶员已构成醉驾")
```

实例：应用海伦公式和余弦定理计算三角形面积

问题：给定三条边的长度，首先判断这三条边是否能够成一个三角形，如果能，计算三角形面积，并判断其构成的是哪种类型的三角形（锐角、直角、钝角）。

海伦公式： $S = \sqrt{p(p - a)(p - b)(p - c)}$ ，其中 $p = \frac{a+b+c}{2}$

余弦定理： $\cos C = \frac{a^2 + b^2 - c^2}{2ab}$

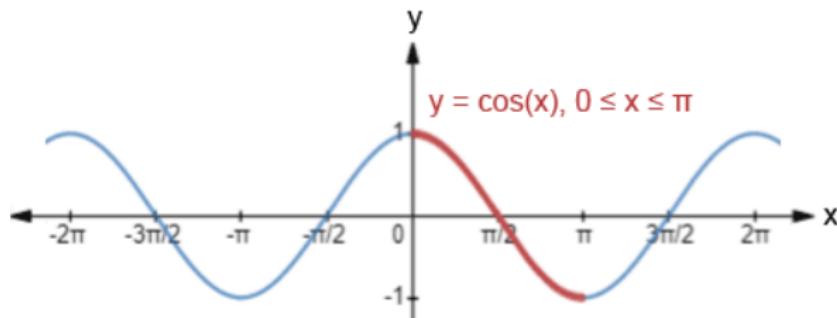


图 15: cosine 函数图像

应用海伦公式和余弦定理的例子 (续)

```
# 将三条边按照长度，从小到大排列
a = 3; b = 12; c = 10
a,b,c = sorted([a,b,c])
# 任意两边之和大于第三边 (a<b<c) , 因此构成三角形
if a>0 and (a + b) > c:
    # 下面用海伦公式用三边求三角形面积
    p = (a + b + c)/2
    area = float((p*(p-a)*(p-b)*(p-c))**0.5)
    print("The area of triangle is {:.2f}".format(area))
# 根据“同一个三角形内，长边对大角”，以及余弦定理，计算最大角的余弦值
cos_C = (a**2 + b**2 - c**2)/(2*a*b)
if cos_C == 0:
    print("Right triangle") # 直角
elif cos_C < 0:
    print("Obtuse triangle") # 钝角
else:
    print("Acute triangle") # 锐角
else:
    print("Invalid lengths")
```

课堂练习: 个人所得税计算

题目: 键盘接收年收入 (单位为元), 根据下图税率, 使用选择结构计算应缴所得额。

级数	全年应纳税所得额	税率 (%)
1	不超过 36000 元的	3
2	超过 36000 元至 144000 元的部分	10
3	超过 144000 元至 300000 元的部分	20
4	超过 300000 元至 420000 元的部分	25
5	超过 420000 元至 660000 元的部分	30
6	超过 660000 元至 960000 元的部分	35
7	超过 960000 元的部分	45

图 16: 个人所得税税率

选择结构总结

单选、双选结构都是多选结构的特殊形式，下面以多选结构为例，总结选择结构的要点：

- **顺序判断：**依次判断 condition1, condition2, ...，是否满足，只要有一个 condition 满足，则不再进行后续判断，并且执行该 condition 对应的语句块 (statements)，如果都不满足执行 else 对应的语句块 (statements0)；
- **运行一次：**运行完 condition 对应的代码块 (statements) 后，立刻离开选择结构，继续执行后续代码块 (other statements)；
- **冒号、缩进、对齐：**一定记得写冒号，冒号下的代码要缩进，同一层级的代码要对齐。

while 循环

```
# 下面为伪代码，不可直接运行
while condition:
    statements

other statements
```

当条件判断表达式 (condition) 返回结果为 `True` 时，执行语句块 (statements)，再次计算 `condition`，如果结果返回 `True`，则继续执行 `statements`，循环往复...，否则 (`False`)，跳出循环，顺序往下运行与 `while` 对齐的后续代码 (other statements)。

注意：① 判断条件 (condition) 必须是一个能返回布尔值 (或等价于布尔值，如 `0,1,None`) 的表达式，如，关系运算和逻辑运算表达式；② 语句块 (statements) 可以包含任意多行代码；③ 选择结构只计算一次 `condition`，`while` 循环计算多次 `condition`，每次运行完语句块 (statements) 就立刻再次判断 `condition` 是否满足。

```
# 笨办法求1+2+3+ ... +100的和
result = 0
number = 1
while number < 101:
    result += number
    number += 1 # number = number + 1
print(result)

# 输出小于10的奇数
a = 1
while a < 10:
    print(a)
    a += 2
print('我就是与while对齐的后续代码^_^')
```

注意: 上述两例中, `while` 循环中的 statements 中都对 condition 中的变量进行了修改, 才使得 condition 可以返回 `False`, 程序跳出 `while` 循环, 从而执行后续代码 (other statements)。

for 循环

```
# 下面为伪代码，不可直接运行
for item in iterables:
    statements
```

通过遍历可迭代对象 (iterables) 中的元素 (item)，来控制语句块 (statements) 执行的次数。典型的可迭代对象有序列 (列表，元组，字符串)、集合、字典、range、zip、enumerate 等。

```
for char in '中华民族伟大复兴':
    print(char)
for char in list('中华民族伟大复兴'):
    print(char)
for char in tuple('中华民族伟大复兴'):
    print(char)
for char in set('中华民族伟大复兴'):
    print(char) # 集合无序
```

内置函数 range

range 和 enumerate 是两个经常与 for 循环搭配使用的内置函数，典型用法如下例：

```
# 打印出 1-9 范围内的数字
for i in range(1,10):
    print(i)
# 以 3 为步长打印出 1-9 范围内的数字
for i in range(1,10,3):
    print(i)
```

range(start, end, step) 返回一个 range 对象，使用内置函数 list 可将该 range 对象转化为列表，列表会包含从 start 到 end-1 之间的整数，步长为 step。此处的 start、end 和 step 与列表切片所使用的参数类似，都是有头没有尾、带步长就跳、正负步长分别对应从左到右和从右到左，对应错误则结果为空。

内置函数 enumerate

```
for idx, num in enumerate(range(48, 59)):  
    print(idx, num)
```

内置函数 `enumerate` 接收一个可迭代对象为参数，返回一个 `enumerate` 对象，通过 `list` 函数可将该对象转化为列表，列表中包含的元素均为形如 `(index, element)` 的二元组，`element` 表示可迭代对象中的一个元素，`index` 为该元素在可迭代对象中的索引 (`index` 本质上是从 0 开始的计数器，看下面例子)。

```
for idx, num in enumerate(set([1, 2, 3])):  
    print(idx, num)
```

集合无序，上例中 `idx` 不表示索引，表示 `num` 是第几个被取出来的。

注意：在使用 `range` 和 `enumerate` 时，不是必须要在其外层套一个 `list` 函数，套 `list` 函数便于查看他们的内容。

课堂练习

题目 1：使用 `for` 循环遍历字典中的键值对。

```
# 题目1
actors = ['沈腾', '马丽', '艾伦']
actor_dict = dict([(101+idx, name) for idx, name in
→ enumerate(actors)])
for idx, name in actor_dict.items():
    print(idx, name)
```

题目 2：计算 1~1000 范围内 3 的倍数和 7 的倍数的数字之和。

题目 3：水仙花数是一个 3 位数，它的每个位上的数字的 3 次幂之和等于它本身（如： $1^3 + 5^3 + 3^3 = 153$ ），求所有水仙花数。

课堂练习-答案

```
# 题目2
sum = 0
for i in range(1, 1001):
    if i % 3 == 0:
        sum += i
    elif i % 7 == 0:
        sum += i
print(sum)
```

```
# 题目3
for i in range(100, 1000):
    sum = 0
    for j in str(i):
        sum += int(j)**3
    if sum == i:
        print(i)
```

`break`、`continue`、`pass`都是 Python 的保留关键字，用法如下：

- `break`：只能出现在循环结构 (`for` 循环, `while` 循环) 中的语句块 (`statements`) 中，程序一旦运行到 `break`，立刻跳出循环，继续执行后续语句 (`other statements`)；
- `continue`：只能出现在循环结构 (`for` 循环, `while` 循环) 中的语句块 (`statements`) 中，程序一旦运行到 `continue`，立刻结束本次循环，进入下次循环；
- `pass`：可以单独出现在任何一行，主要作用是保证代码格式的规范和完整性，程序一旦遇到 `pass`，什么操作也不执行。

注意：① 当 `break` 和 `continue` 出现在嵌套循环中时，只作用于它所在的那一层循环；② `break`、`continue`、`pass` 都是单独出现在某一行，同一行没有其他代码。

51. break,continue,pass, 嵌套

```
vowel = 'aeiou'

# 打印输出 br
for char in 'brown':
    if char in vowel:
        break
    print(char)

# 打印输出 brwn
for char in 'brown':
    if char in vowel:
        continue
    print(char)

# 打印输出 brwn
for char in 'brown':
    if char in vowel:
        pass
    else:
        print(char)
```

题目：对英文句子去做去元音化操作 (Disemvoweling)，即，将单词中非首尾元音字母去除，这是短信 (SMS) 语言的典型特征，阅读起来需要的认知努力很小。

```
vowel = 'aeiou'  
sent = 'The quick brown fox jumps over the lazy dog'  
word_list = []  
  
for word in sent.strip().split():  
    char_list = []  
    for idx,char in enumerate(word):  
        if char in vowel and idx not in [0, len(word) -1]:  
            pass  
        else:  
            char_list.append(char)  
    word_list.append(''.join(char_list))  
print(' '.join(word_list))  
# The qck brwn fx jmps ovr the lzy dg
```

52. 循环结构与 else 子句搭配

```
# 下面为伪代码，不可直接运行
while condition:
    statements1
else:
    statements2
other statements
```

```
# 下面为伪代码，不可直接运行
for item in iterables:
    statements1
else:
    statements2
other statements
```

在循环未经 break 打断的情况下，整个循环完成后，执行 else 子句下的语句块 (statements2)，再执行后续代码 (other statements)。

52. 循环结构与 else 子句搭配

题目：找出 [2, 100] 中的素数。素数（质数），指在大于 1 的自然数中，除了 1 和该数自身外，无法被其他自然数整除的数。

```
for i in range(2, 101):
    for j in range(2, i):
        if i % j == 0:
            break
    else:
        print(i, '是素数')
```

52. 循环结构与 else 子句搭配

题目：寻找整数 x 的最大因子（除 1 和自身外）。

```
x = int(input('Please input a integer bigger than 1: '))

for num in range(x-1, 1, -1):
    if x % num == 0:
        print('The biggest factor is {}'.format(num))
        # print(f'The biggest factor is {num}')
        break
    else:
        print('{} is a prime number'.format(x))
```

题目：求两个整数 a 和 b ($a > 1, b > 1$) 的最大公约数。

```
# a = 8, b = 9; a = 12, b = 16
a = int(input("Please input the first number: "))
b = int(input("Please input the second number: "))
for i in range(min(a,b), 1,-1):
    if a%i == 0 and b%i == 0:
        print("{}和{}的最大公约数是{}".format(a,b,i))
        break
else:
    # 如果两个数的最大公约数是1，则称这两个数互质（互素数）
    print("{}和{}是互素数".format(a,b))
```

题目：最多三次密码输入机会。

```
PASSWORD="123456"
max_times = 3
flag = False
for i in range(3):
    pwd = input("Please input the password: ")
    if pwd == PASSWORD:
        flag = True
        break
    else:
        print("Come on baby, try again, {} times
              left.".format(max_times - i - 1))
if flag:
    print("Great, Baby!")
else:
    print("Sorry, Baby, See you tomorrow!")
```

题目: 1-100 之间不是 7 的倍数的数字之和。

```
sum = 0
for i in range(1, 101):
    if i%7 == 0:
        continue
    else:
        sum += i
print(sum)
```

题目: 思考下面代码的运行结果。

```
i = 0
sum = 0
while i < 5:
    i += 1
    if i == 3:
        continue
    sum += i
print(sum)
```

题目：输出字符串中的每个字符，h 除外。

```
for letter in 'Python':  
    if letter == 'h':  
        pass  
    else:  
        print('Current letter is {}'.format(letter))
```

题目：输出 100 之内的所有素数，每行 10 个依次输出。

```
tmp_list = []
for i in range(2, 100):
    for j in range(2, i):
        if i % j == 0:
            break
    else:
        tmp_list.append(str(i))
    if len(tmp_list) == 10:
        print(', '.join(tmp_list))
        tmp_list.clear()

if len(tmp_list)>0:
    print(', '.join(tmp_list))
```

题目：输出如下图所示的九九乘法表。

1x1=1									
1x2=2	2x2=4								
1x3=3	2x3=6	3x3=9							
1x4=4	2x4=8	3x4=12	4x4=16						
1x5=5	2x5=10	3x5=15	4x5=20	5x5=25					
1x6=6	2x6=12	3x6=18	4x6=24	5x6=30	6x6=36				
1x7=7	2x7=14	3x7=21	4x7=28	5x7=35	6x7=42	7x7=49			
1x8=8	2x8=16	3x8=24	4x8=32	5x8=40	6x8=48	7x8=56	8x8=64		
1x9=9	2x9=18	3x9=27	4x9=36	5x9=45	6x9=54	7x9=63	8x9=72	9x9=81	

```
# 第i行
for i in range(1, 10):
    # 第j列
    for j in range(1, i+1):
        # j x i
        print('{0}x{1}={2}'.format(j, i, i*j), end=' ')
    print() # 换行
```

53. 课堂实训

题目：使用 * 作为填充物打印输出如下图所示的高度为 10 的等腰三角形。

```
row = 10 # 每一行打印两部分，左侧空白和右侧星号
for i in range(row):
    # 打印左侧空白，9,8, ..., 1, 0, (10 - n - 1)
    for _ in range(row - i - 1):
        print(' ', end='')
    # 打印星号，1, 3, 5, ..., 19, (2n + 1)
    for _ in range(2 * i + 1):
        print('*', end='')
    print()
```

题目 1: 用蒙特卡罗法计算圆周率 π 的值。

蒙特卡罗方法是指用随机数通过求解概率而获得近似值的方法。

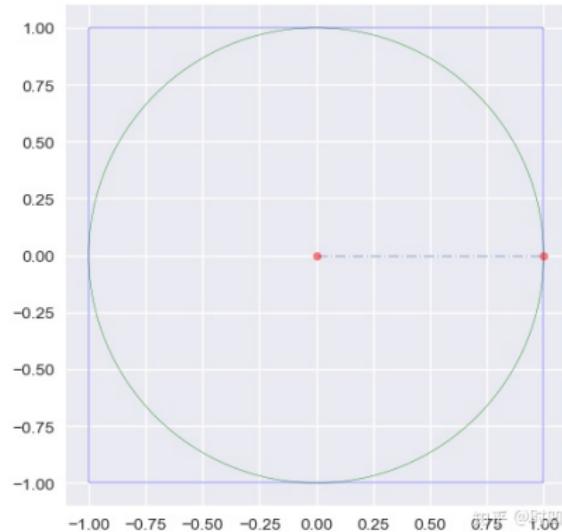


图 17: 以原点为圆心的单位圆

请先思考从上图中随机取一个点，这个点落在单位圆内（包含边界）的概率是多少。

假设圆的半径为 r , 圆的面积为 $S_{circle} = \pi r^2$, 其外接正方形的面积为 $S_{square} = (2r)^2 = 4r^2$

那么, 对外接正方形里的任一个点, 它落在圆里的概率为:

$$P(\text{in circle}) = \frac{S_{circle}}{S_{square}} = \frac{\pi}{4}$$

通过上述公式, 转换得:

$$\pi = 4 \times P(\text{in circle}) \approx 4 \times \frac{m}{n}$$

上式中使用频率近似概率, 从外接正方形中随机采样 n 个点, 其中有 m 个点落在了圆内。

落在圆内的点满足如下条件:

$$(x - x_0)^2 + (y - y_0)^2 \leq r^2, (x_0, y_0) \text{ 为圆心坐标}$$

下面以图17中的单位圆为例，半径为 1，圆心为原点，使用蒙特卡罗法估算圆周率。

```
import math
import random
hit = 0 # 点落在圆内的次数
number_of_trials = 1000000 # 实验次数要足够大
for i in range(number_of_trials):
    # 从外接正方形中随机选择一个点
    x = random.uniform(-1.0,1.0) # 按照均匀分布取横坐标值
    y = random.uniform(-1.0,1.0) # 按照均匀分布取纵坐标值
    # 如果点落在圆内，增加一次计数
    if x**2 + y**2 <= 1:
        hit += 1
print('math.pi is {:>30.8f}'.format(math.pi))
print('The estimated one is
→ {:>20.8f}'.format(4*(hit/number_of_trials)))
```

题目 2: 用微元法计算定积分 $\int_0^1 x^2 dx$ 的值 ($\frac{1}{3}$)。

微元法: 将积分值表示的面积看做无穷多个小矩形的面积之和 (即, 分割、近似、求和)。

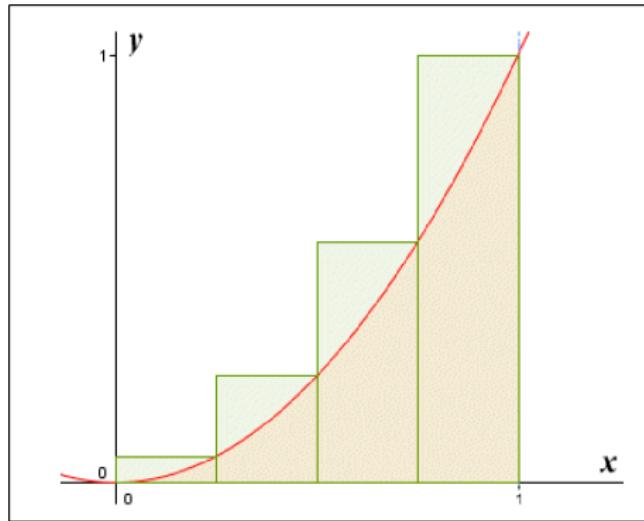


图 18: $f(x) = x^2$ 的函数曲线

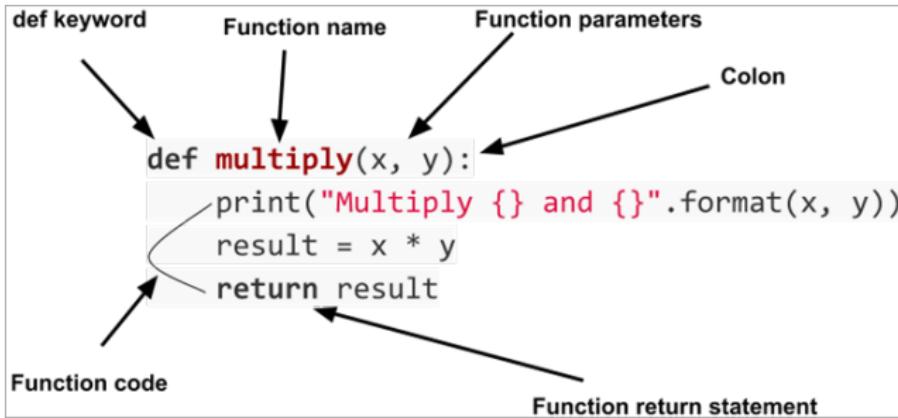
下面使用循环结构, 将上图中函数曲线下方不规则图形分割为足够多个小矩形, 用这些小矩形的面积和近似计算积分值。

```
import math

N = 1000000 # 分割为1000000个小矩形

sum_of_area = 0
i = 1
while i <= N:
    small_area = (1/N) * (i/N)**2 # 宽×高, 求小面积
    sum_of_area += small_area # 小面面积累加
    i += 1
print(sum_of_area)
```

函数: 为实现一个操作或特定功能而集合在一起的语句集，避免代码复制带来的错误或漏洞，不仅可以实现代码的复用，还可以保证代码的一致性。



一个完整的函数由如上图所示的几部分组成：**def** 关键字 (def keyword)，函数名 (function name)，函数参数 (function parameters)，冒号 : (colon)，函数体 (function code)。函数体内 **return** 语句只执行一次，执行完成后，不再运行后续语句。

```
def multiply(x,y):  
    print("Multiply {} and {}".format(x, y))  
    result = x * y  
    return result
```

注意：函数体中的 `return` 语句不是必需的，如果不写，调用函数后返回 `None`，如果所定义的函数不需要参数，可以不写参数，但是括号必需写，看下面例子：

```
def color_print():  
    # 屏幕打印输出不同颜色的字符串，内容为HZNU  
    for i in range(91,96):  
        print("\033["+str(i)+"m"+"HZNU"+"\033[0m")  
  
result = color_print()  
if result == None:  
    print('The call of Function color_print returns None')
```

函数调用

```
# 调用multiply函数，忽略返回值
multiply(3, 5)

# 调用multiply函数，并将函数的返回值复制给变量r
r = multiply(3, 5)

print(r) # 15

# 调用color_print函数，忽略返回值None
color_print()

# 调用color_print函数，将返回值赋值给变量r
# 对于没有返回值的函数，这种调用方式没什么意义
r = color_print()
print(r)
```

注意：① 函数调用可以将返回值赋值给其他变量，也可以忽略返回值；② 函数调用必须写括号，即使函数不需要指定参数。

局部变量: 在函数体内定义的变量，只能在函数内部被访问。

全局变量: 在所有函数之外创建的变量，可以在程序的任意位置访问。

```
# 此处的x, y为全局变量
x, y = 3, 5

def myfunct():
    # 此处的x, y为局部变量
    x, y = 8, 9
    result = x * y

p = myfunct() # 函数调用
# 使用全局变量做计算并将计算结果赋值给全局变量z
z = x + y
print(z) # 8, 不是17
print(p) # None
```

由上例可见，局部变量和全局变量可以同名，但是在函数内部定义的局部变量不能在函数外访问。

形式参数与实际参数

```
# 定义函数
def multiply(x,y):
    print("Multiply {} and {}".format(x, y))
    result = x * y
    return result

# 调用函数
p, q = 8, 9
multiply(p, q)
```

在上例中，定义函数时写的参数 `x` 和 `y` 是形式参数（形参），调用函数时传递给函数的参数 `p` 和 `q`（或者说 8 和 9）叫做实际参数（实参）。

Python 的参数传递

```
# 定义函数
def multiply(x,y):
    print("Multiply {} and {}".format(x, y))
    result = x * y
    return result

# 调用函数

p, q = 8, 9
# 这是引用传递，引用了变量p和q的值8和9
multiply(p, q)
# 这是值传递
multiply(5, 6)
```

注意：引用传递的本质是将变量指向的数据对象传递到函数中，上例中，变量 p 和 q 指向的数据对象为整数 8 和 9。

```
# 定义函数
def my_double(my_obj):
    my_obj *= 2
    return my_obj

a = '000'
b = [1,2]
# 函数调用
my_double(a)
my_double(b)
print(a) # 000
print(b) # [1,2,1,2]
```

上面例子中，变量 `a` 和变量 `b` 分别指向不可变对象（字符串）和可变对象（列表），因此，在函数调用后，变量 `b` 的值发生了变化，变量 `a` 的值不变。所以，当进行引用传递时，要注意，如果变量指向的是可变对象，函数体的代码是可以改变变量值的。

默认参数

定义函数时，可以为参数指定默认值。

如果在调用函数时，未指定参数，则使用参数的默认值作为参数值运行函数；

如果参数不存在默认值，并且调用时也没指定参数值，则程序出错。

```
def my_power(base, exponent=2):  
    return base ** exponent  
  
print(my_power(3)) # 9  
print(my_power()) # Error
```

上例中定义的函数 `my_power`，包含两个参数 `base` 和 `exponent`，参数 `base` 没有默认值，在调用函数时必须为其指定值，参数 `exponent` 有默认值，为 2，调用函数时，如不指定其值，则使用 2 作为其值运行函数。

定义函数时，没有默认值的参数在前，具有默认值的参数在后，否则，程序出错 (SyntaxError)。

```
# Correct
def my_power(base, exponent=2):
    return base ** exponent

# Error
def my_power(exponent=2, base):
    return base ** exponent
```

Python 不支持函数重载，如定义多个重名函数，调用时执行最后一个定义的函数。

函数重载 (方法重载): 是某些编程语言 (如 C++、C#、Java、Swift、Kotlin 等) 具有的一项特性，该特性允许创建多个具有不同实现的同名函数。对重载函数的调用会运行其适用于调用上下文的具体实现，即允许一个函数调用根据上下文执行不同的任务。

按位置传参和按关键字传参

按位置传参: 按照函数定义时的顺序进行参数传递。

按关键字传参: 按照 `name=value` 的格式进行参数传递。

```
# my_sum的三个参数均有默认值
def my_sum(a = 1, b = 2, c = 3):
    return a**0 + b**1 + c**2

# 按位置传参
my_sum(7, 8, 9)
# 按关键字传参
my_sum(c=7, a=8, b=9)
# 按位置传参和按关键字传参混合使用
my_sum(7, 8, c=9)

my_sum(7, b=8, 9) # Error
```

注意: 按位置传参和按关键字传参混合使用时, 位置参数在前, 关键字参数在后, 否则, 程序出错 (`SyntaxError`)。

```
# 按关键字传参，对有默认值的参数使用其默认值  
my_sum(a=9)
```

```
# 按位置传参，对有默认值的参数使用其默认值  
my_sum(7)
```

```
# 按位置传参和按关键字传参混合使用  
my_sum(7, c=8)
```

```
# 按位置和按关键字传参混合使用，并且引用传递和值传递混合使用  
x, y, z = 7, 8, 9  
my_sum(x, b=z, c=8)
```

```
# 按位置传参和按关键字传参混合使用，但是给参数a传递了多个值  
my_sum(7, a=8) # Error
```

可变参数

注意：在定义函数时，不固定参数的数量，调用时也可以使用不同个数的参数，可使用两种参数名来实现：`*args` 和 `**kwargs`，分别对应参数元组（按位置传参）和参数字典（按关键字传参）。

```
def hello_args(para1, *args):
    print("para1 : ", para1)
    print("type(args):", type(args))
    print("args : ", args)
    for idx,arg in enumerate(args):
        print("args{}:".format(idx+1), arg)
```

```
hello_args() # Error, 必须要为para1指定参数值, 因为它没有默认值
hello_args('hello') # 按位置传参, 没有为*args传参
hello_args('hello', 'this', 'is', 'mc.zhang') # 按位置传参
```

上例中，`*args` 用来定义个数不确定的参数元组，在函数体中可以通过 `for` 循环和索引访问参数元组，对于 `*args`，可以传递 0 或多个参数。

```
def hello_args(para1, *args):
    print("para1 :", para1)
    print("type(args):", type(args))
    print("args :", args)
    for idx,arg in enumerate(args):
        print("args{}:".format(idx+1), arg)

args_list = ['this', 'is', 'mc.zhang']
# 先序列(列表)解封, 再按位置传参
hello_args(*args_list)
# 先序列(列表)解封, 再按位置传参
hello_args('hello', *args_list)

# 先序列(列表)解封, 再混合使用按关键字传参和按位置传参, 不行
# 这样就违背了“位置参数在前, 关键字参数在后”的传参规则
hello_args(para1 = 'hello', *args_list) # Error
```

```
def hello_kwargs(para1, **kwargs):
    print("para1 : ", para1)
    print("type(kwargs) : ", type(kwargs))
    print("kwargs : ", kwargs)
    for key, value in kwargs.items():
        print("{0} = {1}".format(key, value))
```

`hello_kwargs()` # Error, 必须要为`para1`指定参数值, 因为它没有默认值

`hello_kwargs('hello')` # 按位置传参, 没有为`*kwargs`传参

按位置传参和按关键字传参混合

```
hello_kwargs("hello", kwarg_1='this', kwarg_2='is',
→ kwarg_3='mc.zhang')
```

上例中, `**kwargs` 用来定义个数不确定的参数字典, 在函数体中可以通过 `for` 循环和关键字 (`key`) 访问参数字典, 对于 `**kwargs`, 可以传递 0 或多个参数。

```
def hello_kwargs(para1, **kwargs):
    print("para1 : ", para1)
    print("type(kwargs): ", type(kwargs))
    print("kwargs:", kwargs)
    for key, value in kwargs.items():
        print("{0} = {1}".format(key, value))

kwargs_dict = {'kwarg_1': 'this', 'kwarg_2': 'is', 'kwarg_3':
    'mc.zhang'}
# 先字典解封，再按位置传参和按关键字传参混合
hello_kwargs("hello", **kwargs_dict)
# 按位置传参，Error，对于**kwargs，只能按关键字传参
hello_kwargs('hello', 'this', 'is', 'mc.zhang')
# 先字典解封，再按关键字传参
hello_kwargs(para1 = "hello", **kwargs_dict)
hello_kwargs(**kwargs_dict, para1 = "hello")
# Error，违背了“位置参数在前，关键字参数在后”的规则
hello_kwargs(**kwargs_dict, "hello")
```

可变参数实例

```
def adder(*num):# 求任意个数字之和
    sum = 0
    for n in num:
        sum = sum + n
    print("Sum:",sum)

adder(3,5)
adder(1,2,3,5,6)

def intro(**data):# 打印通讯录
    print("\nData type of argument:",type(data))
    for key, value in data.items():
        print("{} is {}".format(key,value))

intro(Firstname="Sita", Lastname="Sharma", Age=22,
      Phone=1234567890)
intro(Firstname="John", Lastname="Wood",
      Email="johnwood@nomail.com", Country="Wakanda", Age=25,
      Phone=9876543210)
```

57. 函数的参数

```
def my_max(x,y):
    return x if x > y else y

l = [8,-8]
t = ('good', 'good.')
my_max(*l) # 8
my_max(*t) # 'good.'

def avg_grade(chinese = 80, math = 85):
    return (chinese + math)/2

grade_dict = {'chinese':88, 'math':99}
avg_grade(**grade_dict) # 93.5

def avg_grade_mul(chinese = 80, math = 85, **kwargs):
    return (chinese + math + sum(kwargs.values()))/(2+len(kwargs))

grade_dict = {'english':88,'history':90}
avg_grade_mul(**grade_dict) # 85.75
```

定义函数时不同类型参数的顺序

(位置参数, 默认值参数, 可变参数元组, 命名参数, 可变参数字典)

```
def my_func(a, b, c = 0, *args, d, **kwargs):
    print('a:      {}'.format(a),
          'b:      {}'.format(b),
          'c:      {}'.format(c),
          'args:   {}'.format(args),
          'd:      {}'.format(d),
          'kwargs: {}'.format(kwargs),
          sep='\n')

my_func('a', 'b', 88, *[-1, -2, -3], d='d',
→   **{'name': 'Tom', 'age': '18'})
```

本小节通过一系列函数实例展现函数的功能封装性。

题目：给定一个身份证号，验证其是否合法：① 总体校验；② 校验地区；③ 校验日期。

```
# 校验字符和长度
def verify_char_length(ids):
    if len(ids) != 18:
        return False
    if not ids[:-1].isdigit():
        return False
    if ids[-1] not in '0123456789X':
        return False
    return True
```

```
# 校验第18位是否正确
def verify_last_num(ids):
    ids_17 = ids[:-1]
    weights = [7, 9, 10, 5, 8, 4, 2, 1, 6, 3, 7, 9, 10, 5, 8, 4, 2]
    S = sum([int(num)*weight for num, weight in
             zip(list(ids_17), weights)])
    T = S % 11
    R = (12 - T) % 11
    if R == 10:
        last_num = 'X'
    else:
        last_num = R
    if ids[-1] == str(last_num):
        return True
    else:
        return False
```

```
# 校验前六位地区编码是否正确
def verify_area(ids):
    import _locale
    _locale._getdefaultlocale = (lambda *args: ['zh_CN', 'utf8'])
    with open('./area_dict.json', encoding = 'utf-8') as f:
        area_dict = eval(f.read())
    if ids[:6] not in area_dict.keys():
        return False
    else:
        return True

# 校验闰年
def is_leap_year(year):
    # 公元年份为4的倍数但非100的倍数；公元年份为400的倍数
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
        return True
    else:
        return False
```

```
def verify_date(ids):
    year = int(ids[6:10])
    if year > 2022 or year < 1900:
        return False
    month = int(ids[10:12])
    if month > 12 or month < 1:
        return False
    day = int(ids[12:14])
    if month in [1,3,5,7,8,10,12]:
        if day > 31 or day < 1:
            return False
    elif month in [4,6,9,11]:
        if day > 30 or day < 1:
            return False
    else:
        if is_leap_year(year):
            if day > 29 or day < 1:
                return False
        else:
            if day > 28 or day < 1:
                return False
    return True
```

```
def verify_id(ids):
    if verify_char_length(ids):
        if all([verify_last_num(ids), verify_area(ids), verify_date(id)
               ↪ s])):
            print("VALID")
            return True
        else:
            print("INVALID")
            return False
    else:
        print("INVALID")
        return False

# https://www.dute.org/fake-id-card-number
verify_id(ids = "110113198702121018") # VALID
verify_id(ids = "110113198703121018") # INVALID
```

匿名函数：使用 `lambda` 表达式创建，语法为

`lambda parameters:expression`

- ① 参数间用逗号隔开，允许参数有默认值，表达式的计算结果为匿名函数的返回值，只能由一个表达式组成，不能有其他的复杂结构；
- ② 可将 `lambda` 表达式赋值给一个变量，此变量可作为函数使用；
- ③ 调用 `lambda` 表达式的语法与调用函数完全相同。

```
(lambda x, y:x**y)(2,3) # 直接调用定义的匿名函数, 8
# 使用条件表达式定义匿名函数
my_max = lambda x, y:x if x > y else y
# 定义匿名函数，并将其赋值给变量my_sub
my_sub = lambda x, y=10: x - y
# 调用匿名函数
my_sub(5) # -5
# 使用匿名函数指定sorted函数的排序依据
word_freq = [('my',18), ('go',5), ('can',29), ('exam',3)]
sorted(word_freq, key = lambda item:item[1], reverse=True)
# [('can', 29), ('my', 18), ('go', 5), ('exam', 3)]
```

`map`、`reduce`、`filter` 三个内置函数用于对序列数据进行批量处理，优点是可以使代码更简洁，通常用 `lambda` 表达式（匿名函数）来定义如何处理序列中的元素。

注意：可以使用 `for` 循环等价地实现这三个内置函数的功能。

`map` 使用提供的函数对指定序列做处理，返回对序列中每个元素的处理结果，用法为：

`map(function, iterable, ...)`

```
sent = 'The details of the ICCCT 2023 are as follows'
word_list = sent.split()
m = map(lambda item:len(item), word_list)
type(m) # map
list(m) # [3, 7, 2, 3, 5, 4, 3, 2, 7]
# 计算身份证号校验位的第一步，乘积和
weights = [7, 9, 10, 5, 8, 4, 2, 1, 6, 3, 7, 9, 10, 5, 8, 4, 2]
ids = '110113198702121018'
sum(map(lambda x,y:x*int(y),weights,ids)) # 169
```

reduce 使用提供的函数 (有两个参数) 对指定序列做处理，返回对序列中全部元素的处理结果，先用前两个元素调用函数，用返回结果与第三个元素继续调用函数，用返回结果与第四个元素继续调用函数，用法为：

```
reduce(function, iterable[, init])
```

```
from functools import reduce
reduce(lambda x,y:x*y, range(1,6)) # 5! = 120
reduce(lambda x,y:x*y, range(1,6), 2) # 2*5! = 240
reduce(lambda x,y:max(x,y),[8,3,2,9,10,4]) # 10
```

`filter` 使用提供的函数 (返回布尔值 `True` 或 `False`) 对指定序列的元素进行过滤，用序列中的每个元素调用函数，返回结果中只保留能使得函数返回 `True` 的元素，用法如下：

```
filter(function, iterable)
```

只保留由a-z组成的单词，基础版

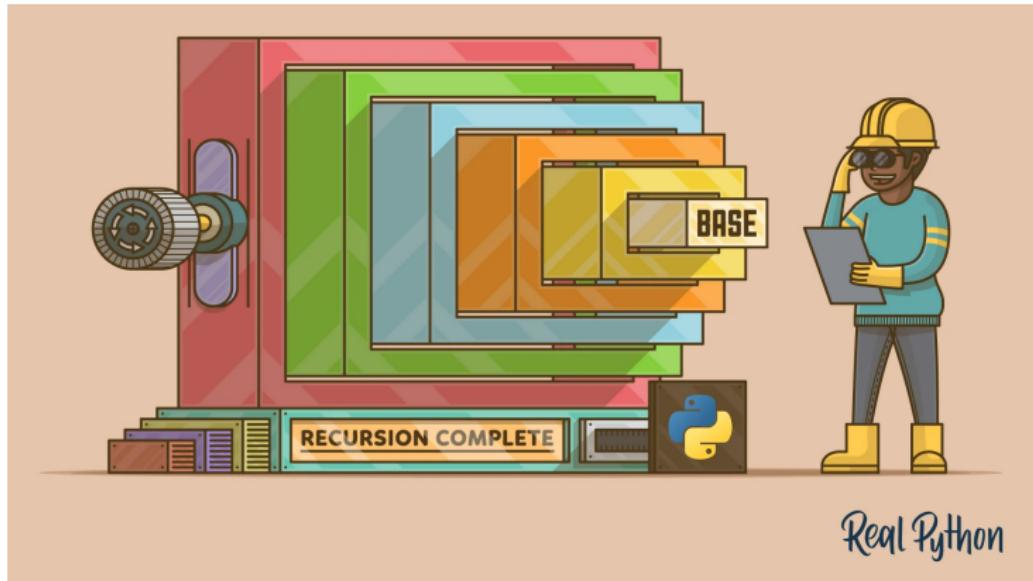
```
word_list = ['python', '^_^', 'time.', '3D']
l = filter(lambda item:item.isalpha(), word_list)
type(l) # filter
list(l) # ['python']
```

只保留由a-z组成的单词，完善版

```
word_list = ['python', '^_^', '我我', 'time.', '3D']
l = filter(lambda item:all([True if ord('a') <=
    ↳ ord(char.lower()) <= ord('z') else False for char in item]),
    ↳ word_list)
list(l) # ['python']
```

递归函数: 在函数体内直接或间接调用自身的函数。

递归思想: 将一个大型的、复杂的问题层层转换为一个与原问题相似的小规模问题来求解，给出一个自然、直观、简单的解决方案。



递归函数的特点

- 使用选择结构将问题分成基础情况和可继续分解情况；
- 有一个或多个基础情况用来结束递归；
- 可继续分解情况通过调用函数自身（递归）被分解为规模更小、与原问题性质相同的子问题；
- 每次递归调用会不断接近基础情况，直到变成基础情况，终止递归。

```
# 求阶乘
def fact(n):
    # if双选结构
    if n == 0: # 只有一种基础情况
        return 1
    else:
        return n * fact(n-1) # 非基础情况递归调用自身
```

斐波那契数列由 0 和 1 开始，之后的斐波那契数就是由之前的两数相加而得出，前几个斐波那契数是：0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987。

```
def fibo(n):
    # 多选结构
    if n == 1: # 基础情况1
        return 0
    elif n == 2: # 基础情况2
        return 1
    else:
        return fibo(n-1) + fibo(n-2) # 非基础情况递归调用自身
```

注意：① 编写递归函数时，需要仔细考虑基本情况；② 一般的递归函数会占用大量的程序栈，尤其是当递归深度特别大的时候，有可能导致栈溢出；③ 当递归不能使全部的问题简化收敛到基本情况时，程序就会无限运行下去并且在程序栈溢出时导致运行时错误（递归函数写错了）；④ 掌握递归必须多练习，常应用。

课堂练习

1. 请定义函数，使用循环结构计算 n 的阶乘。
2. 请定义函数，使用循环结构计算第 n 个斐波那契数。
3. 请定义函数，使用循环结构输出包含前 n 个斐波那契数的列表。
4. 在梵文中，短音节 S 占一个长度单位，长音节 L 占两个长度单位，请定义函数，找出所有可能的长短音节组合方式，使得组合之后的结果长度为 n 。例如 $V_4 = \{LL, SSL, SLS, LSS, SSSS\}$ ， V_4 可进一步划分为两个子集合： $\{LL, LSS\}$ (将 L 与 $V_2 = \{L, SS\}$ 中的每个元素组合可得)， $\{SSL, SLS, SSSS\}$ (将 S 与 $V_3 = \{SL, LS, SSS\}$ 中的每个元素组合可得)。

课堂练习答案

1. 请定义函数，使用循环结构计算 n 的阶乘。

```
def fact(n):
    result = 1
    for i in range(1,n+1):
        result *= i
    return result
```

课堂练习答案

2. 请定义函数，使用循环结构计算第 n 个斐波那契数。

```
def fibo_loop(n):
    if n == 1:
        return 0
    elif n == 2:
        return 1
    else:
        a = 0
        b = 1
        for i in range(n-2):
            c = a + b
            a, b = b, c
    return c
```

课堂练习答案

3. 请定义函数，使用循环结构输出包含前 n 个斐波那契数的列表。

```
def fibo_list_loop(n):
    if n == 1:
        return [0]
    elif n == 2:
        return [0, 1]
    else:
        a = 0
        b = 1
        l = [0, 1]
        for i in range(n-2):
            c = a + b
            a, b = b, c
            l.append(c)
    return l
```

课堂练习答案

4. 请定义函数，找出长度为 n 的梵文长短音组合。

```
def virahanka(n):
    # 第一种基本情况
    if n == 0:
        return []
    # 第二种基本情况
    elif n == 1:
        return ["S"]
    else:
        s = ["S" + prosody for prosody in virahanka(n-1)]
        l = ["L" + prosody for prosody in virahanka(n-2)]
        return s + l
```

61. 递归函数

汉诺塔游戏: 将柱子 A 上的 n 个盘子移动到柱子 C 上, 需遵守如下规则:

- ① 一次只能移动一个圆盘;
- ② 只能移动最顶部的圆盘;
- ③ 大圆盘必须在小圆盘下;

请思考所需次数最少的移动步骤。[点我在线试玩。](#)

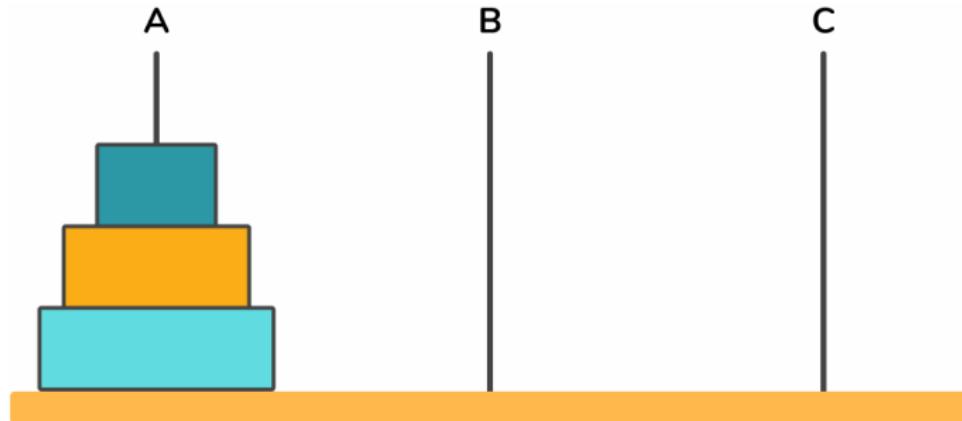


图 19: 3 个盘子的汉诺塔游戏

最小步数的基本思路类似于要把大象装冰箱，总共分三步 ($n \geq 2$):

- ① 把前 $n - 1$ 个盘子放到 B 柱 (把冰箱门打开);
- ② 把第 n 个盘子放到 C 柱 (把大象装冰箱);
- ③ 把前 $n - 1$ 个盘子从 B 柱放到 C 柱 (把冰箱门带上)。

```
def move(n, left, center, right):
    if n == 1:
        print('{} ---> {}'.format(left, right))
    else:
        move(n-1, left, right, center)
        move(1, left, center, right)
        move(n-1, center, left, right)
```

对于 n 个盘子，最少步数为 $2^n - 1 = 2 \times (2^{n-1} - 1) + 1$ ，
 $(2^{n-1} - 1)$ 对应上面的第一步和第三步，1 对应第二步。

生成器是创建可迭代对象的一种方式，其保存数据生成的方式（算法），有助于节省内存，创建生成器的常用方式有两种：

第一种：将列表推导式的边界符由中括号变为圆括号即可；

第二种：定义生成式函数，生成式函数与一般函数的主要区别在于，返回结果使用 `yield` 子句而非 `return` 语句。

```
import sys
g = (i**2 for i in range(1000))
l = [i**2 for i in range(1000)]
sys.getsizeof(g) # 在内存中占112个字节
sys.getsizeof(l) # 在内存中占8856个字节
```

生成器函数返回一个生成器对象，通常使用 `for` 循环依次获得生成器对象的每一个生成值，也可以通过其他内置函数将其转换为某种可迭代对象（如，列表，元组）。下面定义一个生成器函数，返回斐波那契数列的前 n 个值。

```
# 先看课堂练习题2的答案
def fibo_loop(n):
    if n == 1:
        return 0
    if n == 2:
        return 1
    else:
        a = 0
        b = 1
        for i in range(n-2):
            c = a + b
            a, b = b, c
        return c
```

62. 生成器

将上页代码稍作改动即可得到生成前 n 个斐波那契数的生成器函数。

```
def fibo_generator(n):
    if n == 1:
        yield 0
    elif n == 2:
        yield 0
        yield 1
    else:
        yield 0
        yield 1
        a = 0
        b = 1
        for _ in range(n-2):
            c = a + b
            yield c
            a,b = b,c

for item in fibo_generator(3):
    print(item)
fibo_list = list(fibo_generator(3))
```

理解生成器的要点

- 使用 `next` 函数查看生成器的下一个生成值，生成器中的代码运行时，每次遇到 `yield` 子句时函数会暂停并保存当前所有的运行信息，返回 `yield` 子句的生成值，并在下一次执行 `next` 函数时从当前位置继续运行；
- 使用 `for` 循环查看生成器的每一个生成值，相当于多次使用 `next` 函数依次查看生成器的生成值，从第一个看到最后一个；
- 当返回最后一个生成值后，再执行 `next` 函数会报错，使用 `for` 循环遍历生成器的每一个生成值时，遍历完最后一个生成值后，结束循环，不会报错，所以，通常使用 `for` 循环遍历访问生成器的生成值；（看下一页例子）
- 生成器只能前进，不能回看，就是只能依次生成下一个值。（看下一页例子）

```
# 定义一个生成器，赋值给变量g，表达式i**2用于计算生成器每次返回的值
g = (i**2 for i in range(2))
next(g) # 0
next(g) # 1
# 根据定义，生成器g只能生成两个值，所以下述代码会报错
next(g) # Error

# 生成器g已经通过上面两次调用next函数，生成了所有值，
# 此时将其转化为列表，它没有新的元素可生成，所以返回空列表
list(g) # []

g = (i**2 for i in range(3))
# 使用for循环遍历生成器的每一个值
for item in g:
    print(g)

g = (i**2 for i in range(3))
next(g) # 0
list(g) # [1, 4]
list(g) # []
```

```
# 0, 1, 1, 2, 3
# 调用生成器函数，将返回的生成器赋值给变量fibog
fibog = fibo_generator(5)

next(fibog) # 0
next(fibog) # 1
list(fibog) # [1,2,3]
list(fibog) # []
```

63. open 函数

本讲主要学习文本文件的打开和读写操作，暂不涉及操作其他类型的文件（如，图像、音视频）。使用内置函数 `open` 打开文本文件，其用法如下：

```
file_object = open(file_path, open_mode)
```

其中，参数 `file_path` 表示文件路径，没有默认值，必须提供；参数 `open_mode` 表示打开模式，默认值为 `rt` 表示文本只读模式。

```
f = open('./stopwords.txt') # 打开存储停用词表的文件
stopwords_str = f.read() # 读取文件中的全部内容
f.close() # 关闭文件
# 将文件中的内容stopwords_str先去掉一层引号，再作为Python语句运行
stopwords_list = eval(stopwords_str) # 将文件中的内容转化为列表

# with关键字和open配合使用可以避免手动关闭文件
with open('./stopwords.txt') as f:
    stopwords_str = f.read()
stopwords_list = eval(stopwords_str)
```

注意：推荐 `open` 函数和关键字 `with` 配合使用，避免手动关闭文件。

文件路径

绝对路径: 从根目录开始到文件的路径, 如:

/home/zjz/Desktop/stopwords.txt (linux)

C:\Users\zjz\Desktop\stopwords.txt (Windows)

相对路径: 从当前目录到文件的路径, 如:

./stopwords.txt (linux/Windows, 假定当前目录为桌面)

注意: 推荐使用相对路径, 提高代码的可移植性。如果四六级高频词作业我使用绝对路径, 每个同学拿到作业题干后都需要对绝对路径进行修改后才能运行, 这样的代码可移植性太差。

打开模式

打开模式参数包含三部分，格式如下：

r/w/a t/b [+]

第一部分为读写模式，有如下三种选择 (三选一)：

r: 只读 (only read)，是读写模式的默认值；

w: 只写，文件不存在则创建，文件存在，则清空已有内容，写入新内容；

a: 只追加写入，文件不存在则创建，文件存在，则在已有内容之后追加写入新内容。

第二部分为文件格式，有如下两种选择 (二选一)：

t: 文本格式，是文件格式的默认值；

b: 二进制格式，一般用于打开图片、视频等非文本文件。

第三部分的加号表示在原有基础上可读可写 (可选)

常用的打开模式参数有：rt, wt, at, t 是默认文件格式，可以省略，即，r, w, a。

```
# 读取stopwords.txt中保存的通用英语停用词
with open('./stopwords.txt') as f:
    stopwords_str = f.read()

# 新建领域停用词表，写入三个停用词
with open('./domain_stopwords.txt','w') as f:
    f.write('part\nquestions\nexam') # \n表示换行

# 在新建的领域停用词表中，追加写入两个停用词
with open('./domain_stopwords.txt','a') as f:
    f.write('\nminutes\ncollege') # \n表示换行
```

除上例中常用的三种打开模式参数外，我们看几个复杂例子（不常用）。`rb+` 表示以二进制格式打开一个文件，可读可写，若文件不存在不会创建，`wb+`，表示以二进制格式打开一个文件，可读可写，若文件不存在则创建一个。

使用 `open` 函数打开文件后，可用下面三个方法读取文件中的内容：

① `read`，它有一个可选非关键字参数 `size`，指定读取的内容长度，该参数默认值为-1，表示读取文件中的全部内容。

```
# 读取stopwords.txt中的前7个字符，空格和换行符都要计入
# ['i', '\n'
with open('./stopwords.txt') as f:
    stopwords_str = f.read(7) # f.read(size=7)是错误写法,
    ↳ 因为size是非关键字参数
```

```
# 读取stopwords.txt中的全部内容
with open('./stopwords.txt') as f:
    stopwords_str = f.read()
```

② `readline`，读取文件中一行的内容，包括换行符在内，它有一个可选非关键字参数 `size`，指定读取的内容长度，该参数默认值为-1，表示读取一整行内容。

```
# 读取一行内容, ['i', \n
with open('./stopwords.txt') as f:
    line = f.readline()

# 读取一行内容的前2个字符, [
with open('./stopwords.txt') as f:
    line = f.readline(2)

# 指定的长度超过一行的长度, 只返回一行内容, ['i', \n
with open('./stopwords.txt') as f:
    line = f.readline(20)
```

③ **readlines**，读取文件中的全部内容，返回一个字符串列表，文件中的每一行（包括换行符）是列表中的一个元素。

```
# ['part\n', 'question\n', 'exam\n', 'minutes\n', 'college']
with open('./domain_stopwords.txt') as f:
    lines = f.readlines()
```

```
# 使用for循环逐行读取内容（包括每行末尾的换行符）并处理
with open('./domain_stopwords.txt') as f:
    for line in f:
        print(len(line.strip()))
```

注意：read 和 readline 方法返回的是字符串，readlines 方法返回的是字符串列表。

使用 `open` 函数打开文件后，可用下面两个方法把字符串写入文件中：

- ① `write`，把字符串写入到文件中。

```
# 新建领域停用词表，写入三个停用词
content = 'part\nquestions\nexam'
with open('./domain_stopwords.txt','w') as f:
    f.write(content) # \n表示换行

# 在新建的领域停用词表中，追加写入两个停用词
with open('./domain_stopwords.txt','a') as f:
    f.write('\nminutes\ncollege') # \n表示换行
```

- ② `writelines`，将字符串列表写入文件，即，使用空字符作为连接符，将一个字符串列表拼接为一个字符串，再将字符串写入文件。

```
lines = ['part\n', 'question\n', 'exam']
with open('./domain_stopwords.txt', 'w') as f:
    f.writelines(lines)

with open('./domain_stopwords.txt', 'w') as f:
    f.write(''.join(lines))

lines = ['part', 'question', 'exam']
with open('./domain_stopwords.txt', 'w') as f:
    f.write('\n'.join(lines))
```

注意：没有 `writeline` 方法。

总结

- 使用 `with` 关键字搭配 `open` 函数打开文件；
- 读取文件内容，打开模式参数使用默认值；
- 写内容到文件，打开模式参数使用 `w` 或 `a`；
- 使用字符串方法（以及 `join` 方法）搭配文件读写方法进行读写操作；

注意：处理中文文本读写时，指定系统编码格式为 `UTF-8`；同时在 `open` 函数中指定参数 `encoding` 的值为 `UTF-8`。

```
# 指定系统编码格式为UTF-8
import _locale
_locale._getdefaultlocale = (lambda *args: ['zh_CN', 'utf8'])

# open函数指定encoding参数为UTF-8
f = open('./your_file', encoding='UTF-8')
```

打开文件后可以使用 `tell` 方法查看游标 (cursor) 位置，用 `seek` 方法移动游标，这两个方法的单位均为字节 (byte)，一个英文字符占一个字节，一个 UTF-8 编码的中文字符占 3 个字节。

`seek` 方法有两个非关键字参数，`offset` 参数指定移动的字节数，没有默认值，`whence` 参数指定游标从哪个位置开始移动，默认值为 0，表示从文件头开始移动，另外两个可选值为 1 和 2，分别表示从当前位置和文件末尾开始移动。

```
lines = ['大学', 'exam', '英语考试']
with open('./domain_stopwords.txt', 'w', encoding = 'utf-8') as f:
    f.write('\n'.join(lines))

# windows系统中，以二进制 (b)方式打开文本文件，
# 可能会在每个换行符\n前面自动添加一个\r，来表示换行
f = open('./domain_stopwords.txt', 'rb')
f.tell() # 0
f.seek(3) # 3
f.readline() # b'\xe5\xad\xa6\r\n'
type(b'\xe5\xad\xa6\r\n') # bytes
b'\xe5\xad\xa6\r\n'.decode('utf-8') # 学\r\n
```

```
f.tell() # 8
f.seek(3,1) # 11
f.readline() # m\r\n
f.tell() # 14
f.seek(2,2) # 28
f.seek(6,0) # 6
f.readline() # \r\n
f.seek(-5,1) # 3
f.readline().decode('UTF-8') # 学\r\n
# 下面查看文件对象f的一些属性
f.closed # False, 文件处于未关闭状态
f.encoding # utf-8, 文件编码方式为utf-8, 以二进制方式打开文件,
→ 则不存在该属性
f.mode # rb, 文件打开模式为二进制只读
f.name # ./domain_stopwords.txt, 文件路径
f.close() # 关闭文件, 一定要记得关闭文件
f.closed # True, 文件处于关闭状态
```

注意: 使用 `open` 函数打开文件, 文件读写操作完成后, 一定要记得关闭文件, 如果你怕忘记关闭, 推荐 `open` 函数和 `with` 关键字搭配使用。

内置函数 `print` 不仅能够在屏幕打印输出，还可以将内容写入到文件，如下示例：

```
lines = ['part', 'question', 'exam']
with open('./domain_stopwords.txt', 'w') as f:
    for line in lines:
        print(line, end='\n', file=f)

with open('./domain_stopwords.txt', 'r') as f:
    content = f.read() # part\nquestion\nexam\n
```

`print` 函数的 `end` 参数指定结尾字符，默认值为换行符 `\n`.

使用 `input` 函数接收键盘输入，并写入文件。

```
# 给出一个上联，使用input函数接收键盘输入下联，  
→ 并将上下联写入文件duilian.txt  
left_text = '南通州，北通州，南北通州通南北' # 东当铺，西当铺，  
→ 东西当铺当东西  
right_text = input('上联是: {}，请您赐下联: '.format(left_text))  
with open('./duilian.txt', 'w') as f:  
    print('我的上联是: {}'.format(left_text))  
    print(left_text, file = f)  
    print('您的下联是: {}'.format(right_text))  
    print(right_text, file = f)
```

思考：修改上述代码，使对联在文件中竖排显示（提示：请使用 `zip` 和 `for` 循环）。

文件 `grades.txt` 中保存的内容如下图所示，请你编程：

① 读取 `grades.txt` 文件中的内容；

② 解析读取的内容，计算每位同学的平均分；

③ 将计算结果写入 `average.txt` 文件中，第一列为姓名，第二列为平均分。

姓名	语文	数学
张三	88	64
李四	90	78
王五	65	45
赵六	43	66
冯七	62	56
刘八	90	42
阮九	68	36
陆十	86	46

[点我下载 `grades.txt` 文件 \(右键-另存为，即可保存到本地\)。](#)

```
# 指定windows平台下Python运行时的默认编码类型为UTF-8
import _locale
_locale._getdefaultlocale = (lambda *args: ['zh_CN', 'utf8'])
# 打开文件，读取内容
with open('./grades.txt', encoding='utf-8') as f:
    lines = f.readlines()

# 解析内容，计算平均分
new_lines = ['姓名\t平均分']
for line in lines[1:]:
    l = line.strip().split()
    name = l[0]
    avg = sum(map(float, l[1:])) / len(l[1:])
    avg_str = str(avg)
    new_lines.append(name + '\t' + avg_str)

# 将结果写入文件
with open('./average.txt', 'w') as f:
    f.write('\n'.join(new_lines))
```

请将文本文件流水.txt([点我下载](#), 浏览器中右键-另存为)和账面.txt([点我下载](#), 浏览器中右键-另存为)中的内容解析为二维列表。

```
# 指定windows平台下Python运行时的默认编码类型为UTF-8
import _locale
_locale._getdefaultlocale = (lambda *args: ['zh_CN', 'utf8'])

def my_parse(file_path):
    data = []
    with open(file_path, encoding='UTF-8') as f:
        lines = f.readlines()
    for line in lines[1:]:
        line_list = line.strip().split('\t')
        line_list[-1] = float(line_list[-1].replace(',', ''))
        data.append(line_list)
    return data

d1 = my_parse('./流水.txt')
d2 = my_parse('./账面.txt')
```

为便于各类文本文件读写，参照上一讲模块与包的内容，我自己编写了一个文件读写 package，可用于读写多种文本文件，如 txt, xlsx, csv 等，安装使用方法如下：

- ① 点我下载 utils.zip；
- ② 解压 utils.zip 得到 utils 文件夹，将该文件夹放置到 anaconda3\Lib\ 中；
- ③ 开始使用，示例如下

```
# 指定windows平台下Python运行时的默认编码类型为UTF-8
import _locale
_locale._getdefaultlocale = (lambda *args: ['zh_CN', 'utf8'])

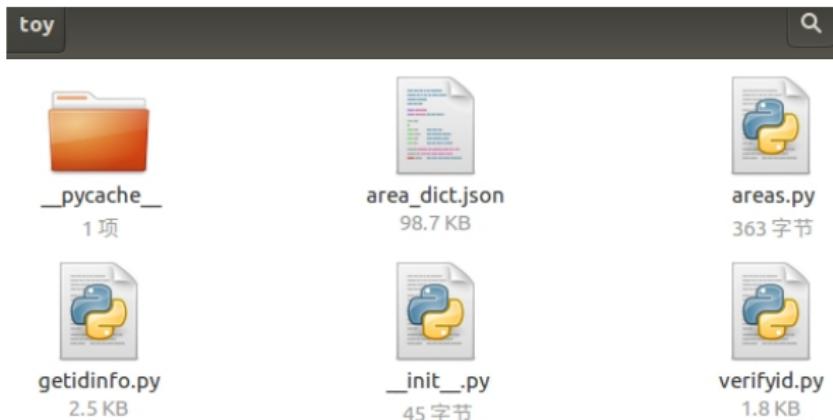
from utils.fileUtils import *

lines1 = readExcelToList('./账面.xlsx')
len(lines1) # 查看行数是否与excel文件中的行数相符
lines1[:3] # 查看内容和格式是否与excel中的相符
help(readExcelToList) # 查看readExcelToList函数用法
```

模块 (module): 把实现相关功能的代码放到一个 `py` 文件中，就是一个模块，其中可以包含类、函数、变量、可执行语句、导入其他模块等。

包 (package): 把多个功能模块放到同一个文件夹构成一个包。

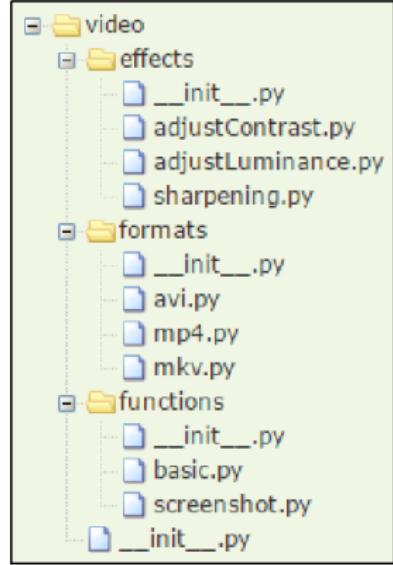
例如，把验证身份证号合法性的代码 (多个函数) 放到名为 `verifyid.py` 的文件中，把获取身份证号信息的代码 (多个函数) 放到名为 `getidinfo.py` 的文件中，就创建了两个名为 `verifyid` 和 `getidinfo` 的模块；把这两个模块 (两个 `py` 文件) 放到名为 `toy` 的文件夹中，就创建了一个名为 `toy` 的文件夹。如下图所示：



69. 模块与包的定义



模块与包的关系



包的结构 (包-子包-模块)

以 Anaconda 为例，Python 的内置（自带）的模块与包的存放路径为 `anaconda3\Lib\`，自己安装的第三方包的路径为 `anaconda3\Lib\site-packages`，也可自己编写包或模块。

使用 `pip` 进行包管理，可在 PyPI 和 GitHub 查找所需第三方包，常用命令（在命令行运行，如需要在 `jupyter` 中运行，则在命令前加上!）如下：

```
pip install <Package Name> # 安装包
pip install <Package Name>==<Version> # 安装包的特定版本
# 使用清华镜像源安装包，速度更快，默认使用国外的镜像，速度较慢
pip install <Package Name> -i
→ https://pypi.tuna.tsinghua.edu.cn/simple
pip show <Package Name> # 查看已安装的包的信息
pip list # 列出已安装的所有包
pip list --outdated # 列出需要更新的包
pip install --upgrade <Package Name> # 升级包
pip uninstall <Package Name> # 卸载包
```

实操：使用清华镜像源安装中文分词包 `jieba`。

Python 导入包或模块 (及其中的内容) 有如下两种方式，其中 something 为“点式结构”，别名是可选的。

第一种: `import something [as alias]`

第二种: `from something import something [as alias]`

包是一种用“点式模块名”构造 Python 模块命名空间的方法，使用“点”连接包结构中各层级的内容 (看下面代码示例)。

```
# 导入包
import toy
# 访问包中的变量，包名 点 模块名 点 变量名
toy.areas.AREA_DICT
# 调用模块中的函数，模块名 点 函数名
toy.verifyid.verify_id('33028220020218410X') # VALID
# 将函数赋值给变量vi，这样调用函数时，就不用写长长的名字了
vi = verifyid.verify_id
vi('999') # INVALID
```

模块的常见属性有 `__doc__` 和 `__name__`，前者为 py 文件头部的注释，用于说明该模块实现的功能、用法等，后者为模块的名字。

```
# 导入包中的一个模块，目标明确，速度快
from toy import verifyid
# 导入包中的所有模块，速度慢
from toy import *
# 导入模块中的一个函数
from toy.verifyid import verify_id
verify_id('999') # INVALID
# 导入模块中的全部内容（所有函数、变量等）
from toy.verifyid import *
# 导入包并起一个别名
import toy as t
# 查看模块的说明文档
print(t.verifyid.__doc__) # check if a ID number is valid.
# 查看模块的名字
print(t.verifyid.__name__) # toy.verifyid
```

注意：上面以 `toy` 包为例的示例代码也适用于导入模块中的内容。

在自己编写包的时候，同一个包内，模块之间互相调用的方式有：

绝对引用： `from toy.areas import area_dict`

相对引用： `from .areas import area_dict`

在本课件配套的 `toy` 包中 `verifyid` 模块中使用了绝对引用方式，调用 `areas` 模块中的变量 `area_dict`，`getidinfo` 模块中使用了相对引用方式，调用 `areas` 模块中的变量 `area_dict`。

72. 异常类型

语法错误: 代码书写不符合 Python 语法, 如选择、循环结构不写冒号。

异常: 代码语法正确, 运行时出现错误, 如, 除 0 错误。

自定义异常: 为便于程序调试, Python 允许用户自定义异常, 并使用关键字 `raise` 主动抛出异常。

```
# SyntaxError, 循环结构不写冒号
for i in range(3)
    print(i)
# ZeroDivisionError, 除0错误
10 % 0
# 自定义异常
class CustomError(Exception):
    def __init__(self, message, status):
        super().__init__(message, status)
        self.message = message
        self.status = status

# 主动抛出自定义异常
raise CustomError('Connected Failed', 404)
```

72. 异常类型

`BaseException` 类是所有异常类的基类，其有四个子类，除 `Exception` 类外，其他三个均为系统级异常，`Exception` 类是所有内置异常类和用户自定义异常类的基类，Python 中异常的分类如下图：

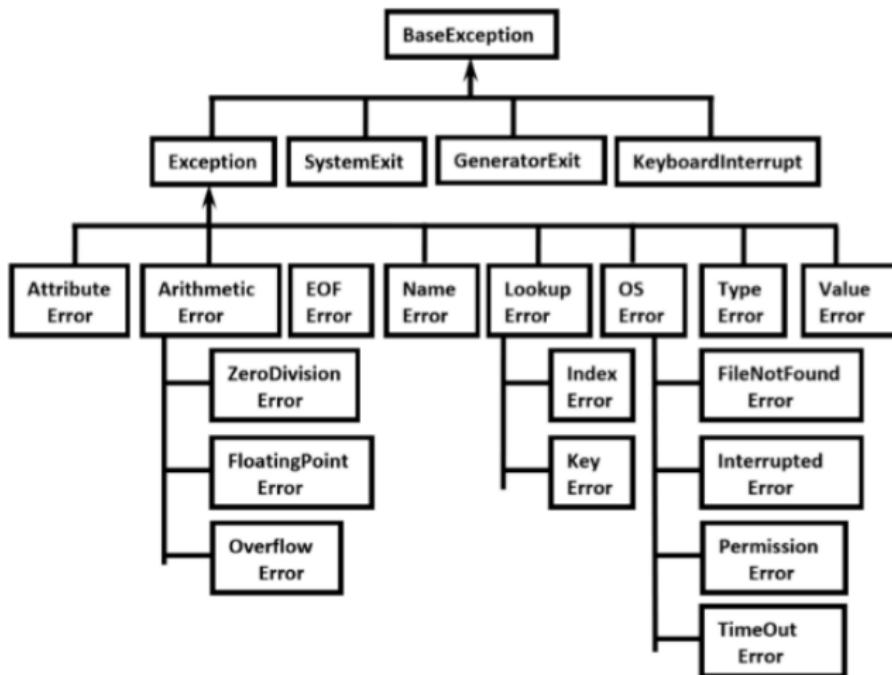


图 20: Python 异常分类

72. 异常类型

Table 1: 常见 Python 内置异常列表

异常名称	描述	异常名称	描述
Exception	普通错误的基类	AttributeError	对象没有这个属性
IOError	输入/输出操作失败	IndexError	序列中没有此索引
KeyError	映射中没有这个键	NameError	未声明/初始化对象
SyntaxError	Python语法错误	SystemError	一般解释器系统错误
ValueError	传入无效参数	ZeroDivisionError	除0异常
ImportError	导入模块异常	TypeError	类型异常
ReferenceError	引用不存在对象异常	AssertionError	assert语句触发的异常

```
4 + spam*3 # NameError: name 'spam' is not defined
'2' + 2 # TypeError: can only concatenate str (not "int") to str
import kkkkk # ModuleNotFoundError: No module named 'kkkkk'
d = {} ; d['abc'] # KeyError: 'abc'
l = [1, 2, 3] ; l[100] # IndexError: list index out of range
s = 'kkk'; s.llower() # AttributeError: 'str' object has no
→   attribute 'llower'
```

使用 **try-except** 语句捕获并处理异常，格式如下：

```
# 下面是伪代码不要直接运行
try:
    statements # 从这些语句中捕获可能的异常
except [ (ErrorType1, ErrorType2, ...) ]:
    statements # 对捕获到的指定异常进行处理
```

```
l = list('Python')
try:
    for i in range(10):
        print(i, l[i], end = ',')
    print('循环顺利结束。')
except IndexError:
    print('\nERROR: 索引超出范围啦...')
```

try 下面的语句块执行过程中发生异常时，则跳过剩余部分，执行 **except** 子句，匹配遇到的异常类型，如匹配成功，则执行 **except** 子句下面的异常处理代码，然后离开 **try-except** 结构，否则程序报错。

`except` 子句可以有多个 (类似于多选结构里的多个条件), `try` 捕获到异常后, 依次匹配每个 `except` 子句后的异常类型, 一旦匹配成功, 就执行相应的异常处理代码, 然后离开 `try-except` 结构。

```
l = list('Python')
try:
    for i in range(10):
        print(i, l[i], end = ',')
    print('循环顺利结束。')
except NameError:
    print('\nERROR: 命名错误... ')
except IndexError:
    print('\nERROR: 索引超出范围啦... ')
    print(10/0)
```

注意: `except` 子句下面的异常处理代码在运行时也可能出现新的异常, 运行上例后, 在处理 `IndexError` 的过程中又引发了 `ZeroDivisionError`。

`except` 后面可以放置多个异常类型 (放在圆括号内, 以逗号分割), 表明若多个异常中至少发生一个, 则执行该部分异常处理代码, 若不放置任何异常类型, 则代表可匹配所有的异常类型。

```
l = list('Python')
try:
    for i in range(10):
        print(i, l[i], end = ', ')
    print('循环顺利结束。')
except (NameError, IndexError):
    print('\nERROR: 命名错误或索引错误')

l = list('Python')
try:
    for i in range(10):
        print(i, l[i], end = ', ')
    print('循环顺利结束。')
except:
    print('\nERROR: There is an error.')
```

73. 异常处理



图 21: 完整的 try-except 结构

try 和 except 必须成对出现，else 子句和 finally 子句是可选的。

下面是使用完整的 try-except 结构的一个示例，其中，except 子句中使用 as 关键字捕获该异常类的示例，便于查看具体异常信息。

```
i = 0
while i < 3:
    try:
        x_input = input("请输入一个数字: ")
        x_int = int(x_input)
        i = 3
    except ValueError as e:
        print("您输入的不是整数，请再次尝试输入！")
        print("具体错误信息如下: {}".format(e))
        i += 1
        if i == 3:
            print('三次机会已经用完，明天再试吧。')
    else:
        print("恭喜你，输入正确！")
    finally:
        print("你的输入为: {}".format(x_input))
```

自定义异常

- Python 允许用户自定义异常，描述内置异常未涉及的异常情况，以便于程序调试；
- 通过定义一个继承 `Exception` 类的派生类来创建自定义异常；
- Python 不会自动抛出或处理任何自定义异常，需要使用 `raise` 语句在合理的场合手工触发异常；
- 在使用自定义异常时，经常需要在捕获异常的同时获取该异常的实例（如上页例子中的 `e`），以获取存储在异常实例中的数据，在 `except` 子句中使用 `as` 关键字加实例名即可。

通过下页实例来理解上面的知识点。

73. 异常处理

```
class MyEnnameError(Exception):
    def __init__(self, value):
        self.value = value
    def __str__(self):
        return str('''English name can only include alphabet letters
        → and space, your input is: {}'''.format(self.value))

import string
i = 3
while i > 0:
    try:
        ename = input("请输入你的英文名: ")
        if set(ename) - set(string.ascii_letters + ' '):
            raise MyEnnameError(ename)
        else:
            break
    except MyEnnameError as e:
        print(e)
        i -= 1
print("你还有{}次输入机会".format(i))
```

断言用于判断一个表达式是否满足，在表达式返回 `False` 时触发 `AssertionError` 异常，显示错误提示信息，语法如下：

```
assert expression [, arguments]
```

断言可以在条件不满足程序运行的情况下直接返回错误，而不必等待程序运行时出现错误，例如，我们的代码只能在满足特点条件的机器上运行时，可以先判断当前及其是否符合条件。

```
import sys
assert ('linux' in sys.platform), "该代码只能在 Linux 下执行"

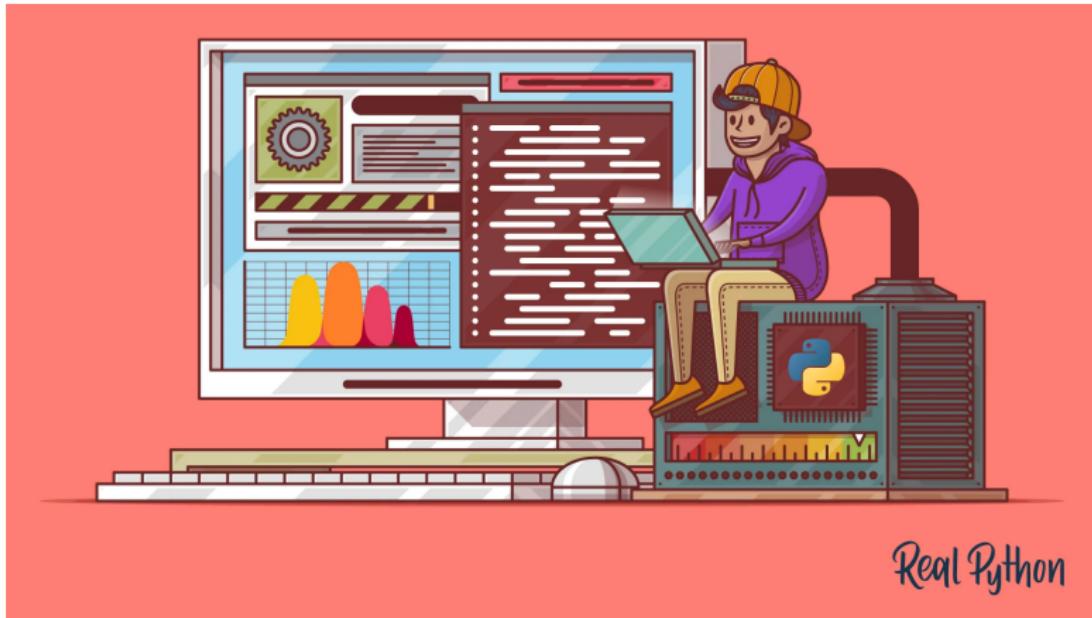
assert 1==1 # 条件为True正常执行，没有指定错误提示信息
print('继续')

salary = -100
assert salary > 0, '工资只能为正数' # 指定了错误提示信息

salary = -100
if not salary > 0:
    raise AssertionError('工资只能为正数')
```

写在课程最后的话

学习这门课程的目的是为了培养计算思维，为今后的学习、工作提供一种高效的工具。



Real Python

这个任务能用 Python 更高效解决吗，如果能，应该如何分解问题，对于分解后的问题，通过查阅互联网寻找解决方法，整合解决方案。

THE END