# 前言



## 获取资源

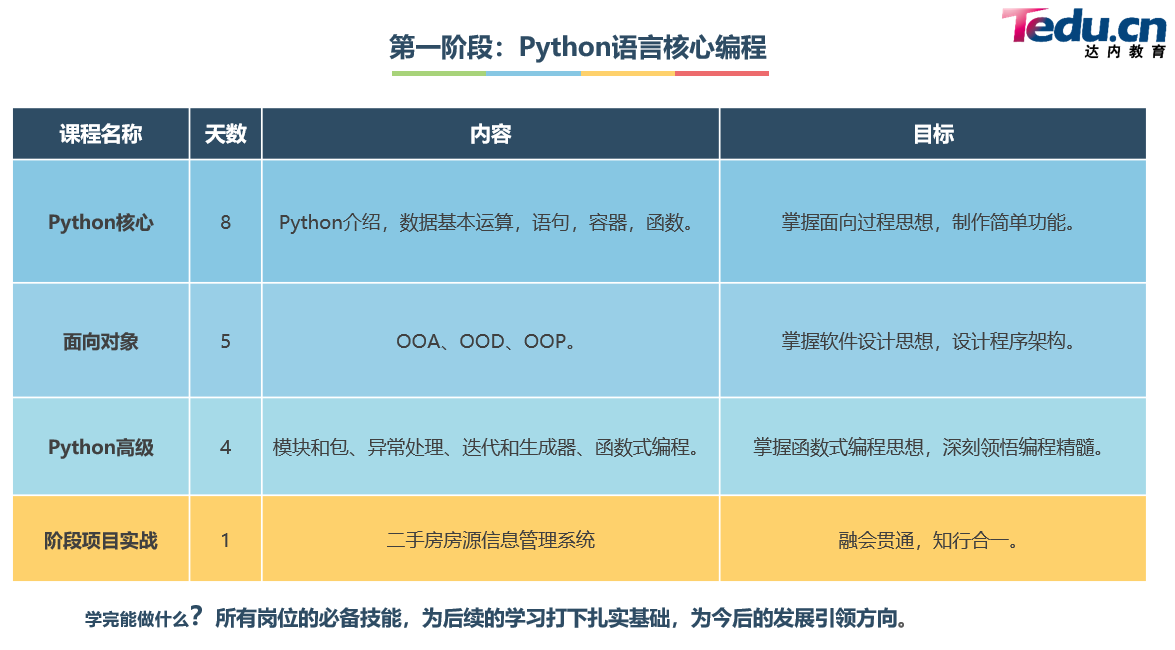
浏览器输入：code.tarena.com.cn

账号：tarenacode

密码：code\_2013

地址： AIDCode/aid2005/01\_month01/to\_student\_for\_month01.zip

## 课程介绍



## 教学理念

**理念一：弱语法，重本质**

是一种弱化语言规则，注重程序原理的学习过程。

语法是表象，只有了解深层机理，才能灵活运用。

学习编程要深入内存，剖析原理，才能看透语法。

就像太极“用意不用力，先在心后在身”的道理。

**理念二：重思想，重设计**

注重解决问题的思维方式，注重编写程序的架构。

通过面向对象三大特征，六大原则学习设计思想。

通过Model View Controller体会程序框架结构。

通过"全国面向对象课程答辩峰会"总结设计思想。

**理念三：是技术，更艺术**

编程是一门技术，但更是一种艺术。

写出高质量的代码（功能性、维护性，灵活性），享受编程所带来的乐趣。

**理念四：项目化，实战化（逻辑理解，串联各个知识点）**

2048核心算法贯穿Python核心。

疫情信息管理系统贯穿面向对象。

集成操作框架贯穿Python高级。

二手房信息管理系统贯穿全阶段。

## 学习方法

**第一步：当天笔记必须整理**

梳理当天所学内容的过程。

防止日后因为遗忘和混淆造成的麻烦。

将理论，代码，图示三合一。

**第二步：当天知识必须理解**

What 是什么，即理解知识点的定义。

Why 为什么，即理解知识点的作用。

Where 在哪里，即理解知识点的适用性。 **最重要的部分，经常用**

How 如何使用，即理解知识点的语法。

**第三步：当天练习必须会做**

课堂演示用眼看 demo01、demo02 …

课堂练习动手敲 exercise01、exercise02 …

独立完成才算会做。

# Python 简介

## 计算机基础结构

### 硬件

1944年，美籍匈牙利数学家冯·诺依曼提出计算机基本结构。

五大组成部分：运算器、控制器、存储器、输入设备、输出设备。

-- 运算器：按照程序中的指令，对数据进行加工处理。

-- 控制器：根据程序需求，指挥计算机的各个部件协调工作。

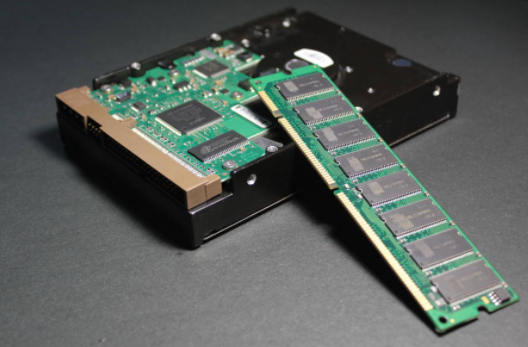
通常将运算器和控制器集成在中央处理器（CPU）中。



-- 存储器：保存各类程序的数据信息。

内存RAM -- 容量小，速度快，临时存储数据

硬盘HDD -- 容量大，速度慢，永久存储数据



输入设备：外界向计算机传送信息的装置。

例如：鼠标、键盘、扫描仪…

输出设备：计算机向外界传送信息的装置。

例如：显示器、音响、打印机…



### 软件



操作系统：

-- 管理和控制计算机软件与硬件资源的程序。

-- 隔离不同硬件的差异，使软件开发简单化。

-- Windows，Linux，Unix。

应用软件：为了某种特定的用途而被开发的软件。

软件：程序 + 文档。

-- 程序是一组计算机能识别和执行的指令集合。

-- 文档是为了便于了解程序所需的说明性资料。

## 基础知识

### Python 定义

是一个免费、开源、跨平台、动态、面向对象的编程语言。

### Python程序的执行方式

#### 交互式 基于测试的

在命令行输入指令，回车即可得到结果。

1. 打开终端
2. 进入交互式：python3
3. 编写代码：print(“hello world”)  **成对打括号和引号，左键调整位置**
4. 离开交互式：exit()

#print("你好 世界!")

#### 文件式

将指令编写到.py文件，可以重复运行程序。

1. 编写文件
2. 打开终端
3. 进入程序所在目录：cd 目录
4. 执行程序： python3 文件名

### Linux常用命令

1. pwd：查看当前工作目录的路径
2. cd：改变工作目录（进入到某个目录）

练习：

1. 在指定目录创建python文件.

--目录：/home/tarena/month01

--文件名：exercise01.py

2. 在文件中写入：print(“你好，世界!”)

3. 运行python程序

### 执行过程\*\*\*\*\*

计算机只能识别机器码(1010)，不能识别源代码(python)。

1. 由源代码转变成机器码的过程分成两类：编译和解释。
2. 编译：在程序运行之前，通过编译器将源代码变成机器码，例如：C语言。

-- 优点：运行速度快

-- 缺点：开发效率低，不能跨平台。

1. 解释：在程序运行之时，通过解释器对程序逐行翻译，然后执行。例如Javascript

-- 优点：开发效率高，可以跨平台；

-- 缺点：运行速度慢。

1. python是解释型语言，但为了提高运行速度，使用了一种编译的方法。编译之后得到pyc文件，存储了字节码（特定于Python的表现形式，不是机器码）。

源代码 -- 编译 --> 字节码 -- 解释 --> 机器码

**.py .pyc 01**

**(只有导入的模块)**

|————1次———|

**第1次编译为了 快，第2次解释为了跨操作平台**

**main.py -- 入口(代码简单,导入其他功能模块)**

**usl.py -- 界面(处理界面逻辑View)**

**bll.py -- 业务(业务逻辑Controller)**

**model.py -- 模型(封装Model)**

### 解释器类型

1. CPython（C语言开发)
2. Jython (java开发)
3. IronPython (.net开发)

Exercise 03

"""

汇率转换器

"""

# 1. 获取数据

usd = input("请输入美元：")

# 2. 逻辑处理

rmb = float(usd) \* 7.1465

# 3. 显示结果

print(usd + "美元=" + str(rmb) + "人民币")

# 书写程序的大致顺序

# 从上到下，从右到左

# 数据基本运算

## 基础知识

### pycharm常用快捷键\*\*\*

1. 移动到本行开头：home键
2. 移动到本行末尾：end键盘
3. 注释代码：ctrl + /
4. 复制行：ctrl +d
5. 删除行：shift + delete
6. 选择列：shift + alt +鼠标左键
7. 移动行：shift + alt + 上下箭头（shift + crtl + 上下箭头）
8. 智能提示：ctrl + space
9. 代码格式化：ctrl+alt+l

### 注释

给人看的，通常是对代码的描述信息。

1. 单行注释：以#号开头。
2. 多行注释：三引号开头，三引号结尾。  **三引号所见即所得**

### 函数

表示一个功能，函数定义者是提供功能的人，函数调用者是使用功能的人。

例如：

1. print(数据) 作用：将括号中的内容显示在控制台中

"""

print 函数

练习:exercise01

"""

# 1. 字面意思：打印(需要打印的内容)

# 2. 作用：向终端输出/显示内容

# 3. 适用性：显示结果

print("你好")

print("世界")

1. 变量 = input(“需要显示的内容”) 作用：将用户输入的内容赋值给变量

"""

input 函数

练习：exercise02

"""

# 　赋值号=：将右边的结果,复制给左边.

# 1. 字面意思：结果 = 输入(提示信息)

# 2. 语法：结果 = input(提示信息)

# 3. 作用：从终端中获取用户输入的信息

# 4. 适用性：获取数据

name = input("请输入姓名：")

# + 连接多个文本

图片包含 游戏机, 物体, 钟表

描述已自动生成print("您好:" + name)

## 变量

1. 定义：关联一个对象的标识符。 **关联-箭头 对象-数据地址 标识符-变量名**
2. 命名：必须是字母或下划线开头，后跟字母、数字、下划线。

不能使用关键字(蓝色)，否则发生语法错误：SyntaxError: invalid syntax。

1. 建议命名：字母小写，多个单词以下划线隔开。

class\_name = “xxx”

1. 赋值：创建一个变量或改变一个变量关联的数据。
2. 语法：

变量名 = 数据

变量名1 = 变量名2 = 数据

变量名1, 变量名2, = 数据1, 数据2

"""

问题：

程序运行在哪里？ -- 内存

程序在处理什么？ -- 数据

变量

定义：在内存中，操作数据。

语法：

变量名 = 数据

练习:exercise03

"""

# name01 得到的是数据("刘陈方")地址

name01 = "刘陈方"

name02 = "闫艺锋"

name01 = "朱林旭"

print("您好：" + name01)# 您好：朱林旭

# name03得到的是变量name02存储的数据("闫艺锋")地址

手机屏幕截图

描述已自动生成手机屏幕截图

描述已自动生成name03 = name02

"""

变量其他写法

删除变量

练习：exercise04

"""

# 写法1：变量名 = 数据

data01 = "悟空"

# 写法2：变量名1, 变量名2 = 数据1, 数据2

data02, data03 = "八戒", "唐僧"

print(data02) # "八戒"

print(data03) # "唐僧"

# 写法3：变量名1 = 变量名2 = 数据

data04 = data05 = "沙僧"

data01 = "大圣"

del data02 # 删除变量data02,数据"八戒"引用计数为0所有被销毁

手机屏幕截图

描述已自动生成手机屏幕截图

描述已自动生成del data04 # 删除变量data04,数据"沙僧"引用计数为1

"""

变量交换

a, b = b, a

"""

# 变量交换通用思想

# bridegroom\_name = "武大郎"

# bride\_name = "潘金莲"

# temp = bridegroom\_name

# bridegroom\_name = bride\_name

# bride\_name = temp

# print("交换后的新郎：" + bridegroom\_name) #

# print("交换后的新娘：" + bride\_name) #

# a = "武大郎"

# b = "潘金莲"

# c = a

# a = b

# b = c

# print(a) #

# print(b) #

a = "武大郎"

b = "潘金莲"

a, b = b, a

print(a) #

手机屏幕的截图

描述已自动生成手机屏幕截图

描述已自动生成print(b) #

## del 语句\*\*\*\*\*

1. 语法:

del 变量名1, 变量名2

1. 作用：

用于删除变量,同时解除与对象的关联.如果可能则释放对象。

1. 自动化内存管理的引用计数：\*\*\*\*\*

每个对象记录被变量绑定(引用)的数量,当为0时被销毁。

引用计数:每个对象记录被变量绑定(引用)的数量,当为0时被销毁。

缺点:循环引用 - 两个垃圾数据互相引用

时机:时刻

标记清除:"全盘"扫描内存,检查(标记)不再使用的数据.（扫描循环引用数据， 引用计数为0 的已经在引用计数中标记，不需要扫描，提高效率）

缺点:效率太低

时机:内存告急

分代回收:将内存分为小中大三代

每次创建新数据都在0代分配空间

当标记清除后,将前一代有用的数据升级到下一代.

前一代所有数据清空

为什么设计三种方法？每一个方法都是弥补前一代的缺点。

实际应用：优化内存

尽量减少产生垃圾/自定义对象池/控制内存管理系统：

尽量减少产生垃圾：使用可变代替不可变容器

自定义对象池： 对象池:每当创建对象时,在池中判断是否存在相同对象,

如果有直接返回地址

如果没有再分配空间创建对象

例如,字符串池/整数池/小数池.....

价值:提高内存利用率

控制内存管理系统：设置标记清除，分代回收的参数

"""

# 循环引用

list01 = []

list02 = []

list01.append(list02)

list02.append(list01)

手机屏幕的截图

描述已自动生成del list01, list02

手机屏幕截图

描述已自动生成

# 产生垃圾的常用代码:

a = "数据1"

a = "数据2" # 当变量a又指向"数据2"时,"数据1"成为垃圾.

b = "数据3"

del b # 因为变量b被销毁,所以"数据3"成为垃圾

# 根据某些逻辑,循环拼接字符串.(频繁修改字符串)

str\_result = ""

for number in range(10):

# 两个不可变对象,相加后会产生新对象

str\_result = str\_result + str(number)

print(str\_result)

# 解决:使用可变对象代替不可变对象

list\_result = []

for number in range(10):

list\_result.append(str(number))

str\_result = "".join(list\_result)

print(str\_result)

#对象池

data01 = "悟空"

data02 = "悟空"

print(id(data01))

print(id(data02))

## 核心数据类型

**在python中变量没有类型，但关联的对象有类型。**

"""

变量没有类型,但是关联的对象有类型.

data = "10"

data = 10

data = 10.0

"""

### 整形int

1. 表示整数，包含正数、负数、0。

如： -5, 100, 0

1. 字面值：

十进制：每位用十种状态计数，逢十进一，写法是0~9。

# -- 十进制(DEC)：每位用十种状态计数，逢十进一，写法是0~9

number01 = 20

二进制：每位用二种状态计数，逢二进一，写法是0b开头，后跟0或者1。

# -- 二进制(BIN)：每位用二种状态计数，逢二进一，写法是0b开头，后跟0或者1。

number02 = 0b10

八进制：每位用八种状态计数，逢八进一，写法是0o开头，后跟0~7。

# -- 八进制(OCT)：每位用八种状态计数，逢八进一，写法是0o开头，后跟0~7。

# 0 1 2 3 ...7 10

number03 = 0o20

十六进制：每位用十六种状态计数，逢十六进一，

写法是0x开头，后跟0~9,A~F,a~f

# -- 十六进制(HEX)：每位用十六种状态计数，逢十六进一，写法是0x开头，后跟0~9 a(10)~f(15)

number04 = 0x10

### 浮点型float

1. 表示小数，包含正数、负数，0.0。
2. 字面值：

小数：1.0 2.5

科学计数法：e/E (正负号) 指数

1.23e-2 (等同于0.0123)

1.23456e5(等同于123456.0)

# -- 小数写法

number05 = 1.5

# -- 科学计数法

# 15 × 10 -1

number06 = 15e-1

print(0.00001) # 1e-05

### 字符串str

是用来记录文本信息(文字信息)。

字面值：双引号

**# 字符串类型str:表达文本**

name = "18"

### 布尔bool

用来表示真和假的类型

True 表示真(条件满足或成立)，本质是1

False 表示假(条件不满足或不成立)，本质是0

"""

bool运算

bool类型 真的True 假的False

表达命题(一个带有判断性质的陈述句)

运算符

比较运算符

相同== 不同!=

(数值)大小 > >= < <=

练习：exercise01

"""

# 命题：我是个总统(输入的职业等于总统)

# result = input("请输入您的职业：") == "总统"

# print(result)# True False

# 命题：您成年了(输入的年龄是大于18岁)

result = int(input("请输入您的年龄：")) >= 18

print(result)

## 数据类型转换

1. 转换为整形: int(数据)
2. 转换为浮点型:float(数据)
3. 转换为字符串:str(数据)
4. 转换为布尔:bool(数据)

结果为False：bool(0) bool(0.0) bool(None)

1. 混合类型自动升级：

1 + 2.14 返回的结果是 3.14

1 + 3.0 返回结果是: 4.0

"""

类型转换

结果 = 数据类型(待转换数据)

"""

# 案例：计算明年年龄

# str\_age = input("请输入你的年龄：")

# int\_age = int(str\_age) + 1

# print("明年你" + str(int\_age))

# str --> int

number01 = int("18") # 18

# float -(截断删除)-> int

number02 = int(1.9) # 1

print(number02)

# int --> str

str01 = str(18)# "18"

# float --> str

str02 = str(1.5) #"1.5"

# int --> float

float01 = float(18) # 18.0

# str --> float

float02 = float("1.5") # 1.5

# 注意：

**# 字符串转换为其他类型时,必须长得像目标类型. ”18” “18.5”**

# str --> ?

## 运算符

### 算术运算符

+ 加法

- 减法

\* 乘法

/ 除法：结果为浮点数

// 整除：除的结果去掉小数部分

% 求余

\*\* 幂运算

优先级从高到低： ()

\*\*

\* / % //

+ -

"""

算数运算符 + - \* 幂运算\*\*

小数商/ 整数商// 余数%

"""

number01 = 5

number02 = 2

#

print(number01 + number02) # 7

# 因为字符串与数字不能拼接，所以将数字转换为字符串

print("结果是:" + str(number01 + number02)) # 7

# 5 的 2次方 5 \* 5 --> 25

print(number01 \*\* number02)

print(number01 / number02) # 5 / 2 --> 2.5

print(number01 % number02) # 5 % 2 --> 1

print(number01 // number02) # 5 // 2 --> 2

### 增强运算符

y += x 相当于 y = y + x

y -= x 相当于 y = y - x

y \*= x 相当于 y = y \* x

y /= x 相当于 y = y / x

y //= x 相当于 y = y // x

y %= x 相当于 y = y % x

y \*\*= x 相当于 y = y \*\* x

"""

增强运算符 += -= \*= //= /= %= \*\*=

在算数运算符基础上增加了对自身赋值的功能

"""

# number01 = 10

# number01 + 5 # 算数运算符

# print(number01) # 10

number01 = 10

# number01 = number01 + 5 # 算数运算符

# 自增

number01 += 5 # 算数运算符

print(number01) # 15

### 比较运算符

< 小于

<= 小于等于

> 大于

>= 大于等于

== 等于

!= 不等于

返回布尔类型的值

比较运算的数学表示方式:0 <= x <= 100

### 逻辑运算符

#### 与and

表示并且的关系，一假俱假。

示例:

True and True # True

True and False # False

False and True # False

False and False # False

#### 或or

表示或者的关系，一真俱真

示例:

True or True # True

True or False # True

False or True # True

False or False # False

#### 非 not

表示取反

例如：

not True # 返回False

not False # 返回True

"""

逻辑运算符

判断多个命题关系

与and -- > 并且

或 --> 或者

非 --> 取反

语法：

命题1 and 命题2 都满足

命题1 or 命题2 一个就行

not 命题1 取反

练习：exercise02

"""

# 来自丈母娘的灵魂质问：

# 1. 有钱 并且 有才

# int(input("请输入你的财产：")) > 10000000

# input("请问您是否有才华：") == "有"

# result = int(input("请输入你的财产：")) > 10000000 and input("请问您是否有才华：") == "有"

# print(result)

# 2. 有钱 或者 有才

# result = int(input("请输入你的财产：")) > 10000000 or input("请问您是否有才华：") == "有"

# print(result)

# 3. 非 not

print(not True) # False

绿色的标志

描述已自动生成

#### 短路运算

一但结果确定，后面的语句将不再执行。

### 身份运算符

语法:

x is y

x is not y

作用：

is 用于判断两个对象是否是同一个对象,是时返回True,否则返回False。

is not 的作用与is相反

### 优先级

高到低：

算数运算符

比较运算符

增强运算符

身份运算符

逻辑运算符

# 语句

## 行

1. 物理行：程序员编写代码的行。
2. 逻辑行：python解释器需要执行的指令。
3. 建议一个逻辑行在一个物理行上。
4. 如果一个物理行中使用多个逻辑行，需要使用分号；隔开。

5. 如果逻辑行过长，可以使用隐式换行或显式换行。

隐式换行：所有括号的内容换行,称为隐式换行

括号包括: () [] {} 三种

显式换行：通过折行符 \ (反斜杠)换行，必须放在一行的末尾，目的是告诉解释器,下一行也是本行的语句。

"""

行

"""

# 3 个物理行 1 个逻辑行(命令)

result = 1 + 2 \

+ 3 + \

4 + 5

result = (1 + 2

+ 3 +

4 + 5)

# 1 个物理行 3 个逻辑行(命令)

a = 1;b = 2;c = a + b

## 选择语句

### If elif else 语句

1. 作用:

让程序根据条件选择性的执行语句。

1. 语法:

if 条件1:

语句块1

elif 条件2:

语句块2

else:

语句块3

1. 说明:

elif 子句可以有0个或多个。

else 子句可以有0个或1个，且只能放在if语句的最后。

"""

选择语句

有选择性的执行语句

写法1：

如果 条件:

满足条件执行的代码

否则:

不满足条件执行的代码

练习：exercise03

"""

if input("请问您是否有才华：") == "有":

print("嫁给你")

else: # 互斥

print("拒绝你")

"""

选择语句

if 条件1:

满足条件1执行的语句

if 条件2:

满足条件2执行的语句

else:

不满足条件2执行的语句

写法2：

if 条件1:

满足条件1执行的语句

elif 条件2:

不满足条件1,但满足条件2执行的语句

else:

以上条件都不满足

调试Debug

让程序中断,逐语句执行,审查程序执行过程以及

变量的取值( 找错误的过程 )

步骤：

1. (在可能出错的行)加断点

2. 开始调试Debug

3. (待断点命中后)按下F8(执行下一句)

.....

4. 关闭程序ctrl + shift + f4

练习:exercise04~

修改名称的快捷键：shift + F6

"""

number = int(input("请输入数字："))

if number > 0:

print("正数")

elif number < 0:

print("负数")

else:

print("零")

### if 语句的真值表达式

if 100:

print("真值")

等同于

if bool(100):

print("真值")

"""

bool 类型转换

if 语句真值表达式

"""

# 为false的值： 0 0.0 "" -> 空None .... vs (" ")->输出空格

result = bool(None)

print(result)

message = input("请输入：")

# if message != "": # 输入的不是空

# if message != 0:

if message: # bool(message) 输入的有值

print("您输入的是:" + message)

else:

print("您没有输入")

### 条件表达式

语法：变量 = 结果1 if 条件 else 结果2

作用：根据条件(True/False) 来决定返回结果1还是结果2。

"""

条件表达式

根据条件给变量赋值

"""

# if input("请输入性别:") == "男":

# value = 1

# else:

# value = 0

value = 1 if input("请输入性别:") == "男" else 0

print("性别编号：" + str(value))

## 循环语句

"""

循环对比：（适用性）

while 循环:根据条件重复执行

　　　　　　例如：纸张对折到珠穆拉玛峰

for 循环：重复获取容器元素

for+range：根据次数重复执行 **#range（5）代表循环5次，不需要计数变量count参与**

　　例如：累加１－１００之间数字

练习:exercise09

"""

### while语句

1. 作用:

可以让一段代码满足条件，重复执行。

1. 语法:

while 条件:

满足条件执行的语句

else:

不满足条件执行的语句

1. 说明:

else子句可以省略。

在循环体内用break终止循环时,else子句不执行。

"""

while 循环

语法

while 条件：

满足条件执行的循环体

写法1：延长程序的生命周期

while True:

循环体

if 条件:

break

练习:exercise01

"""

# si循环

while True:

num = int(input("请输入电梯承载人数:"))

weight = float(input("请输入电梯承载重量:"))

if num <= 10 and weight <= 1000:

print("电梯正常运行")

else:

print("电梯超载")

if input("请输入q键退出：") == "q":

break # 跳出循环

"""

写法2：循环计数

三大要素

循环变量 - 开始值

循环条件 - 结束值

增减变量 - 间隔

练习:exercise02

"""

# 打印结果：0 1 2 3 4

count = 0 # 计数器 1

while count < 5: # 2 4 6 ..

print(count) # 3 5 7 .**.**

count += 1

**#循环体内print放置在count计数之前或之后时 需调整开始值，代码对比如下**

count = 0

while count < 4:

print(count)

count += 1

""代码对比"""

count = -1

while count < 4:

count += 1

print(count)

"""

累计运算

步骤：

循环前 -- 创建

循环中 -- 累加

循环后 -- 结果

练习:exercise03~05

"""

# 打印结果：1 2 3 4 5

# 运算：1 + 2 + 3 + 4 + 5

# 核心思想： 之前 累加 当前

# sum\_value += count

sum\_value = 0 # 之前

count = 1

while count < 6:

sum\_value += count

count += 1 # 当前

print(sum\_value)

"""

猜数字游戏2.0版本

增加最多猜3次的功能

超过次数提示：游戏失败

猜对了提示：对了,总共猜了x次

while 条件:

循环体

else:

循环条件不满足，才执行

从循环体中的break退出,不执行.

"""

import random

random\_number = random.randint(1, 100) （**包含结束值，特殊**）

count = 0

while count < 3:

count += 1

input\_number = int(input("你猜是哪个数字:"))

if input\_number > random\_number:

print("大了")

elif input\_number < random\_number:

print("小了")

else:

print("对了,总共猜了" + str(count) + "次")

break

else:

print("游戏失败")

### for 语句

1. 作用:

用来遍历可迭代对象的数据元素。

可迭代对象是指能依次获取数据元素的对象，例如：容器类型。

1. 语法:

for 变量列表 in 可迭代对象:

语句块1

else:

语句块2

1. 说明:

else子句可以省略。

在循环体内用break终止循环时,else子句不执行。

"""

for 循环

for 变量 in 容器:

变量存储的就是容器中记录的数据

练习:exercise06

"""

message = "我是齐天大圣"

for item in message:

手机屏幕截图

描述已自动生成 print(item)

# for item in 100:

# print(item)

"""

for for 嵌套

**1. 在思想上讲，从内到外编写**

**2. 在执行过程，外层执行一次内层执行多次**

**3. 在行列控制，外层决定行内层决定列**

练习:exercise01~03

"""

"""

print("老王",end = " ")

print("老王",end = " ")

print("老王",end = " ")

print("老王",end = " ")

print("老王",end = " ")

# 换行

print()

print("老王",end = " ")

print("老王",end = " ")

print("老王",end = " ")

print("老王",end = " ")

print("老王",end = " ")

# 换行

print()

"""

**# 外层循环控制行**

**for r in range(2): # 0 1**

**# 内层循环控制列**

**for c in range(5): # 01234 01234**

print("老王", end=" ")

print()

"""

**自定义排序算法**

**练习:exercise03**

**"""**

**# 核心思想：**

**# 确定第一个元素是最大值**

**# 确定第二个元素是最大值**

**# ...**

**# 确定第倒数第二个元素是最大值**

# 步骤：

# 1.取出前几个数据(不要最后一个)

# 2.与后面元素进行比较

# 3.发现更xx则交换(取出的 比较的)

list01 = [43, 15, 5, 67, 87, 9]

for r in range(len(list01) - 1): # 0 1

for c in range(r + 1, len(list01)):

if list01[r] > list01[c]:

list01[r], list01[c] = list01[c], list01[r]

print(list01)

### range 函数

1. 作用:

用来创建一个生成一系列整数的可迭代对象(也叫整数序列生成器)。

1. 语法:

range(开始点，结束点，间隔)

1. 说明:

函数返回的可迭代对象可以用for取出其中的元素

返回的数字不包含结束点（**对比random记忆**）

（**# import random**

**# # 包含开始结束**

**# random.randint(1,10)**）

开始点默认为0

间隔默认值为1

"""

for + range

整数生成器　range

字面意思：范围

价值：生成一个范围内的整数

练习:exercise07

"""

# 写法１：range(开始,结束,间隔)

# 不包含结束值

for number in range(0, 5, 1): # 0 1 2 3 4

print(number)

# 写法2：range(开始,结束)

# 间隔 默认为１

for number in range(0, 5): # 0 1 2 3 4

print(number)

# 写法3：range(结束)

# 开始 默认为0

for number in range(5): # 0 1 2 3 4

print(number)

## 跳转语句

"""

Python核心2

语句

"""

# def func01():

# for r in range(5): # 执行一次

# for c in range(4): # 执行多次

# # continue # 跳过当次循环

# # break 跳出1层循环

# return # 结束函数(跳出所有循环)

### break 语句

1. 跳出循环体，后面的代码不再执行。
2. 可以让while语句的else部分不执行。

### continue 语句

跳过本次，继续下次循环。

"""

continue

练习:exercise08

"""

# 需求：累加1--100之间所有整数

# sum\_value = 0

# for number in range(1, 101):

# sum\_value += number

# print(sum\_value)

# 要求：能被3整除的数字

# 思想：满足条件，累加.

# sum\_value = 0

# for number in range(1, 101):

# if number % 3 == 0:

# sum\_value += number

# print(sum\_value)

# 思想：不满足条件，跳过.

sum\_value = 0

for number in range(1, 101):

if number % 3 != 0:

continue # 跳过

sum\_value += number

print(sum\_value)

# 容器类型

## 通用操作

### 数学运算符

1. +：用于拼接两个容器
2. +=：用原容器与右侧容器拼接,并重新绑定变量
3. \*：重复生成容器元素
4. \*=：用原容器生成重复元素, 并重新绑定变量
5. < <= > >= == !=：依次比较两个容器中元素,一但不同则返回比较结果。

### 成员运算符

1. 语法：

数据 in 序列

数据 not in 序列

1. 作用：

如果在指定的序列中找到值，返回bool类型。

"""

通用操作

数学运算

成员运算

"""

# 1.

# + 用于【拼接】两个容器

name = "悟空"

name += "八戒"

print(name) # "悟空八戒"

# \* 【重复】生成容器元素

name \*= 3

print(name)

# 比较 依次比较两个容器中元素,一但不同则返回比较结果。（比的是Unicode码，应用少）

print("悟空" =="八戒")

print("悟空" !="八戒")

# 2.

# 元素　在　容器中

# 注意：顺序、连续

print("悟空" in "我是花果山水帘洞孙悟空")

# 元素　在　容器中

print("悟空" not in "我是花果山水帘洞孙悟空")

### 索引index

1. 作用：定位单个容器元素。
2. 语法：容器[整数]
3. 说明：

正向索引从0开始，第二个索引为1，最后一个为len(s)-1。

反向索引从-1开始,-1代表最后一个,-2代表倒数第二个,以此类推,第一个是-len(s)。

"""

**索引Index**

**容器名[整数]**

练习:exercise01

"""

message = "我是花果山水帘洞美猴王齐天大圣"

print(message[0]) # 打印第一个元素

print(message[2]) # 打印第三个元素

print(message[-2]) # 打印倒数第二个元素

# print(message[100])# IndexError: string index out of range

# print(message[-100])# IndexError: string index out of range

print(message[-len(message)]) # 第一个

print(message[len(message) - 1]) # 最后一个

### 切片slice

1. 作用：

定位多个容器元素。

1. 语法：

容器[开始索引:结束索引:步长]

1. 说明：

结束索引不包含该位置元素

步长是切片每次获取完当前元素后移动的偏移量

开始、结束和步长都可以省略

"""

**切片**

**容器名[开始:结束:间隔] 不包含结束索引的元素**

**容器名[开始:结束] 间隔默认为１**

**容器名[:结束] 开始默认为0**

**容器名[:] 开始默认为0**

**容器名[::-1] 从尾到头**

"""

# for number in range( 开始 ,结束 ,间隔 ):

# for number in range( 开始 ,结束):

# for number in range(结束):

message = "我是花果山水帘洞美猴王齐天大圣"

print(message[0:3:1]) # 我是花

print(message[0:8:2]) # 我花山帘

print(message[0:3]) # 我是花

print(message[:3]) # 我是花

print(message[:]) # 我是花果山水帘洞美猴王齐天大圣

print(message[3:]) # 果山水帘洞美猴王齐天大圣

print(message[3:-1]) # 果山水帘洞美猴王齐天大

print(message[::-1]) #圣大天齐王猴美洞帘水山果花是我

print(message[2:250]) # 花果山水帘洞美猴王齐天大圣

print(message[2:7:-1]) #空

print(message[7:2]) #空

### 内建函数

1. len(x) 返回序列的长度
2. max(x) 返回序列的最大值元素
3. min(x) 返回序列的最小值元素
4. sum(x) 返回序列中所有元素的和**(元素必须是数值类型)**

## 字符串 str

### 定义

由一系列字符组成的不可变序列容器，存储的是字符的编码值。

### 编码

1. 字节byte：计算机最小存储单位，等于8 位bit.
2. 字符：单个的数字，文字与符号。
3. 字符集(码表)：存储字符与二进制序列的对应关系。
4. 编码：将字符转换为对应的二进制序列的过程。
5. 解码：将二进制序列转换为对应的字符的过程。
6. 编码方式：

--ASCII编码：包含英文、数字等字符，每个字符1个字节。

--GBK编码：兼容ASCII编码，包含21003个中文；英文1个字节，汉字2个字节。

--Unicode字符集：国际统一编码，旧字符集每个字符2字节，新字符集4字节。

--UTF-8编码：Unicode的存储与传输方式，英文1字节，中文3字节。

#### 相关函数

1. ord(字符串):返回该字符串的Unicode码。
2. chr(整数):返回该整数对应的字符串。

"""

字符串

练习:exercise10

"""

name = "悟空"

name = "孙悟空"

# 1.不可变

# 现象：

# 创建了新字符串"孙悟空",替换了变量name存储的地址

# 原因：

# 因为如果在原有基础上修改，可能破坏其他数据的内存空间。

# 所以"损人利己"的事,不能干。

print(name) # 孙悟空

# 2. 编码

# https://unicode-table.com/cn/#4E00

# 字 --> 数

print(ord("一")) # 19968

print(chr(19968)) # "一"

### 字面值

#### 单引和双引号的区别

1. 单引号内的双引号不算结束符
2. 双引号内的单引号不算结束符

#### 三引号作用

1. 换行会自动转换为换行符\n
2. 三引号内可以包含单引号和双引号
3. 作为文档字符串

#### 转义字符

1. 改变字符的原始含义。

手机屏幕截图

描述已自动生成\’ \” \””” \n(换行) \\ \t \0 空字符

1. 原始字符串：取消转义。r”

a = r”C:\newfile\test.py”

"""

字符串字面值

"""

# 1. 写法

name01 = "悟空" # (推荐)

name02 = '悟空'

# 可见即所得

name03 = """

悟

空"""

name04 = '''悟空'''

# 2.

# 单引号内的双引号不算结束符

# 双引号内的单引号不算结束符

message = '我是"孙悟空"。'

message = "我是'孙悟空'。"

# 3. 转义字符:改变原始含义的特殊字符

# \" \' 换行\n \\

message = "我是\"孙悟空\"。"

print(message)# 我是"孙悟空"。

message = "我是\n孙悟空。"

print(message)#

#### 字符串格式化

1. 定义：

生成一定格式的字符串。

1. 语法：

字符串%(变量)

"我的名字是%s,年龄是%s" % (name, age)

1. 类型码：

%s 字符串 %d整数（%.2d） %f 浮点数（%.2f）

"""

字符串格式化

"格式" % (变量)

在格式中可以写占位符(%s %d %f)

"""

# 灵活的数据

name = "唐僧"

age = 2

score = 100.51

# 按照固定的格式显示

# 字符串拼接（格式复杂代码可读性差）

# print("我是" + name + "今年" + str(age)

# + "考试" + str(score) + "分")

# 占位符%s

print("我是%s今年%.2d考试%.1f分" % (name, age, score))

usd = "15"

rmb = float(usd) \* 7.1465

# print(usd + "美元 =" + str(rmb) + "人民币")

print("%s美元 = %.2f人民币"%(usd,rmb))

## 列表 list

### 定义

列表定义：由一系列**变量**组成的**可变**序列容器。

字符串定义：由一系列字符组成的不可变序列容器。

### 基础操作

1. 创建列表：

列表名 = []

列表名 = list(可迭代对象)

1. 添加元素：

列表名.append(元素)

列表.insert(索引，元素)

"""

**列表基础操作**

**列表创建**

**列表添加**

练习:exercise02

"""

# 1. 创建

# 列表名称 = [元素１,元素２,元素３]

list\_names = ["谭锦岳", "杨淮靖", "张昆鹏"]

# 列表名称 = list(其他容器) 可迭代对象，如字符串 源于类型转换类似int str

list01 = list("孙悟空")

print(list01)

[“孙”“悟”“空”] 列表可以修改，然后再转回原类型

# 2. 添加

# -- 追加

# 列表名称.append(元素)

list\_names.append("王韵璇")

# -- 插入

# 列表名称.insert(需要插入的索引,元素) 索引数可以超过元素总数，自动放到最后一项 (0-n)

list\_names.insert(0,"张帆")

定位元素：

列表名[索引] = 元素

变量 = 列表名[索引]

变量 = 列表名[切片] # 赋值给变量的是切片所创建的新列表

列表名[切片] = 容器 # 右侧必须是可迭代对象，左侧切片没有创建新列表。遍历列表：

正向：

for 变量名 in 列表名:

变量名就是元素

反向：

for 索引名 in range(len(列表名)-1,-1,-1):

列表名[索引名]就是元素

"""

**列表基础操作**

**定位**

**读取**

**修改**

**练习:exercise03**

"""

list\_names = ["谭锦岳", "杨淮靖", "张昆鹏"]

# 1. 索引

item = list\_names[0] # 读取第一个元素

list\_names[0] = "tjy" # 修改第一个元素

# 2. 切片

# 通过切片读取元素时会创建新列表

items = list\_names[1:] # 读取后两个元素

# 通过切片修改元素时会遍历右侧数据,依次存入左侧定位的区域

list\_names[1:] = ["yhj", "zkp"] # 修改后两个元素

# list\_names[1:] = "行吗" # ['tjy', '行', '吗']

# 左侧切片1个位置 右侧４个数据

list\_names[1:1] = ["a", "b", "c", "d"] # ['tjy', 'a', 'b', 'c', 'd', 'yhj', 'zkp']

# 左侧切片全部位置 右侧０个数据

# list\_names[:] = []# 列表没有元素

print(list\_names)

"""

**列表基础操作**

**遍历**

**练习:exercise04**

"""

list\_names = ["谭锦岳", "杨淮靖", "张昆鹏", "佩琪"]

# 1. 从头到尾　获取所有元素

for item in list\_names:

print(item)

# 修改的是循环遍历item,并不是列表中的元素,因此列表不变

# for item in list\_names:

# item = ""

# 2. 通过索引(for + range)定位

# -- 顺序 0 1 2 3

for i in range(len(list\_names)):

if len(list\_names[i]) > 2:

list\_names[i] = ""

print(list\_names)

# -- 倒序 3 2 1 0

# range(开始,结束,间隔) 不包含结束值

# 最大索引（总数-1） 得到0（-1） 倒着（-1）

# range(总数-1, -1, -1)

for i in range(len(list\_names) - 1, -1, -1): #

print(list\_names[i])

1. 删除元素：

列表名.remove(元素)

del 列表名[索引或切片]

"""

**列表基础操作**

**删除**

"""

list\_names = ["谭锦岳", "杨淮靖", "张昆鹏"]

# 1. 根据元素删除

# 列表名.remove(元素)

list\_names.remove("杨淮靖")

# 删除不存在的元素，会报错。

# 建议先判断,如果存在再删除.

if "老张" in list\_names:

list\_names.remove("老张")

# 2. 根据定位删除

del list\_names[0]

del list\_names[:]

print(list\_names)

1. **列表内存图手机屏幕截图

   描述已自动生成**

list03切片list03创建一个新的列表

list04索引获取第一个元素记录的数据地址

"""

列表内存分配

"""

list01 = [10, 20, 30]

# 变量list02得到的是变量list01存储的列表地址（**只有一份列表**）

list02 = list01 **赋值**

# 变量list03得到的是新列表的地址（**复制一份新列表（拷贝），两份列表，一份改了，另一份不变**）

list03 = list01[:]

# 变量list04得到的是第一个列表的第一个元素记录的数据(10)地址

list04 = list01[0]

### 深拷贝和浅拷贝

浅拷贝：复制过程中,只复制一层变量,不会复制深层变量绑定的对象的复制过程。

深拷贝：复制整个依懒的变量。

手机屏幕截图

描述已自动生成

"""

**浅拷贝内存图**

"""

list01 = [10, [20, 30], [40, 50]]

# list02 得到的是变量list01存储的列表地址

list02 = list01 **赋值**

# 通过list02修改列表第一个元素,再通过list01访问受影响

list02[0] = "十"

list02[1][0] = "二十"

# list03 得到的列表浅拷贝(一层)出来的新列表

list03 = list01[:] **浅拷贝**

# 修改第一层(不影响拷贝前的数据list01)

list03[0] = "a"

# 修改第二层(影响拷贝前的数据list01)

手机屏幕截图

描述已自动生成list03[1][1] = "b"

"""

**深拷贝**

"""

# 准备一个拷贝工具

import copy

list01 = [10, [20, 30], [40, 50]]

list02 = list01 # 赋值

list03 = list01[:] # 浅拷贝

list04 = copy.deepcopy(list01)# 深拷贝 **（互不影响）**

### 列表与字符串转换

将多个字符串拼接为一个。

result = "连接符".join(列表)

将一个字符串拆分为多个。

列表 = “a-b-c-d”.split(“分隔符”)

"""

**列表 --> 字符串**

**将多个字符串拼接为一个列表。**

**result = "连接符".join(列表)**

练习：exercise01

"""

list01 = ["孙悟空", "猪八戒", "唐僧"]

# 孙悟空\_猪八戒\_唐僧

result = "\_".join(list01)

print(result)

# 需求：根据xxx逻辑拼接字符串

# 缺点：每次拼接+ 都会产生新字符串对象(之前的数据就成为了垃圾)

手机屏幕的截图

描述已自动生成# 解决方案核心思想：使用可变对象代替不可变对象

# result = ""

# for number in range(10):

# + 产生新字符串

# result = result + str(number)

# print(result)

list\_result = []

for number in range(10):

# append 在原有列表上追加

list\_result.append(str(number))

str\_result = "".join(list\_result)

print(str\_result)

"""

**字符串 --> 列表**

**将一个字符串拆分为多个。**

**列表 = “a-b-c-d”.split(“分隔符”)**

练习:exercise02

"""

# 用一个字符串存储多个信息

# 可以使用split分割多个信息

list\_result = "孙悟空\_猪八戒\_唐僧".split("\_")

print(list\_result)

### 列表推导式

1. 定义：

使用简易方法，将可迭代对象转换为列表。

1. 语法：

变量 = [表达式 for 变量 in 可迭代对象]

变量 = [表达式 for 变量 in 可迭代对象 if 条件]

1. 说明:

如果if真值表达式的布尔值为False,则可迭代对象生成的数据将被丢弃。

"""

**列表推导式**

**根据可迭代对象构建列表时**

**变量 = [表达式 for 变量 in 可迭代对象 if 条件]**

**练习:exercise03**

"""

list01 = [34, 45, 5, 65, 76, 8]

# 快捷键：iter + 回车

# list\_result = []

# for number in list01:

# if number > 10:

# list\_result.append(number)

# print(list\_result)

list\_result = [number for number in list01 if number > 10]

# list\_result = []

# for number in list01:

# list\_result.append(number % 10)

# print(list\_result)

list\_result = [number % 10 for number in list01]

print(list\_result)

#### 列表推导式嵌套

1. 语法：

变量 = [表达式 for 变量1 in 可迭代对象1 for 变量2 in可迭代对象2]

1. 传统写法：

result = []

for r in ["a", "b", "c"]:

for c in ["A", "B", "C"]:

result.append(r + c)

1. 推导式写法：

result = [r + c for r in list01 for c in list02]

"""

**列表推导式 嵌套**

**练习:exercise04**

"""

list01 = ["香蕉", "苹果", "哈密瓜"]

list02 = ["雪碧", "可乐", "牛奶", "绿茶"]

# list\_result = []

# for r in list01:

# for c in list02:

# list\_result.append(r + c)

# print(list\_result)

list\_result = [r + c for r in list01 for c in list02]

## 元组 tuple

### 定义

1. 由一系列变量组成的**不可变**序列容器。 Vs **列表可变**
2. 不可变是指一但创建，不可以再添加/删除/修改**元素**。

### 基础操作

1. 创建空元组：

元组名 = ()

元组名 = tuple()

1. 创建非空元组：

元组名 = (20,)

元组名 = (1, 2, 3)

元组名 = 100,200,300

元组名 = tuple(可迭代对象)

1. 获取元素：

变量 = 元组名[索引]

变量 = 元组名[切片] # 赋值给变量的是切片所创建的新列表

1. 遍历元组：

正向：

for 变量名 in 列表名:

变量名就是元素

反向：

for 索引名 in range(len(列表名)-1,-1,-1):

一些文字和图片的手机截图

描述已自动生成 元组名[索引名]就是元素

"""

**元组 tuple**

练习:exercise04,05

"""

# 1. 创建

# -- 通过元素创建

tuple01 = (10, 20, 30)

# -- 通过其他容器创建

list01 = [40, 50]

tuple02 = tuple(list01)

# 2. 定位

# -- 索引

print(tuple01[-1])

# -- 切片

print(tuple01[-2:])

# 3. 遍历

for item in tuple01:

print(item)

for i in range(len(tuple01) - 1, -1, -1):

print(tuple01[i])

# 4. 特殊

# -- 如果元组只有１个元素,必须写逗号

tuple03 = (50,)

# -- 可以省略小括号

# tuple04 = (50,60,70)

tuple04 = 50, 60, 70

# -- 拆包

# a, b = (80, 90)

# a, b = [80, 90]

# a, b = "悟空"

a, \*b = (80, 90, 100)

print(a) # 80

print(b) # [90, 100]

### 作用

1. 元组与列表都可以存储一系列变量，由于列表会预留内存空间，所以可以增加元素。
2. 元组会按需分配内存，所以如果变量数量固定，建议使用元组，因为占用空间更小。
3. 应用：

变量交换的本质就是创建元组：x, y = （y, x ）

手机屏幕的截图

描述已自动生成格式化字符串的本质就是创建元祖："姓名:%s, 年龄:%d" % ("tarena", 15)

## 字典 dict

### 定义

1. 由一系列键值对组成的可变散列容器。
2. 散列：对键进行哈希运算，确定在内存中的存储位置，每条数据存储无先后顺序。
3. 键必须惟一且不可变(字符串/数字/元组)，值没有限制。

### 基础操作

1. 创建字典：

字典名 = {键1：值1，键2：值2}

字典名 = dict (可迭代对象)

1. 添加/修改元素：

语法:

字典名[键] = 数据

说明:

键不存在，创建记录。

键存在，修改值。

1. 获取元素：

变量 = 字典名[键] # 没有键则错误

1. 遍历字典：

for 键名 in 字典名:

字典名[键名]

for 键名,值名 in 字典名.items():

语句

1. 删除元素：

del 字典名[键]

地图上有字

描述已自动生成

"""

字典dict

练习:exercise06

"""

# 1. 创建

# --使用元素创建

dict01 = {"qtx": 100000, "wk": 100000, "bj": 200000}

# --使用其他容器(该容器内的元素必须能够一分二)

list01 = [("唐僧", 50000), ["猪八戒", 60000], "沙僧"]

# {'唐僧': 50000, '猪八戒': 60000, '沙': '僧'}

dict02 = dict(list01)

print(dict02)

# 2. 添加(字典不存在该key) 字典[键] = 值

if "ss" not in dict01:

dict01["ss"] = 700000

# 3. 定位 字典名[键]

# -- 读取

print(dict01["wk"])

# -- 修改

if "qtx" in dict01:

dict01["qtx"] = 500000

# 4. 删除 del 字典名[键]

del dict01["bj"]

# 5. 遍历

# -- 所有键

# for 键名 in 字典:

for key in dict01:

print(key)

# -- 所有值

# for 值名 in 字典.values():

for value in dict01.values():

print(value)

# -- 所有键和值

# for 键,值 in 字典.items():

# 不建议

# for item in dict01.items():

# print(item[0])

# print(item[1])

for key, value in dict01.items():

print(key)

print(value)

### 字典推导式

1. 定义：

使用简易方法，将可迭代对象转换为字典。

1. 语法:

{键:值 for 变量 in 可迭代对象}

{键:值 for 变量 in 可迭代对象 if 条件}

"""

**字典推导式**

"""

# 　需求：range(1,11)中的数字作为key,平方作为value

# dict01 = {}

# for number in range(1, 11):

# dict01[number] = number \*\* 2

dict01 = {number: number \*\* 2

for number in range(1, 11)

}

# dict01 = {}

# for number in range(1, 11):

# if number % 2 ==0:

# dict01[number] = number \*\* 2

dict01 = {number: number \*\* 2

for number in range(1, 11)

if number % 2 == 0}

print(dict01)

## 集合 set

### 定义

1. 由一系列不重复的不可变类型变量(元组/数/字符串)组成的可变散列容器。
2. 相当于只有键没有值的字典(键则是集合的数据)。

### 基础操作

1. 创建空集合：

集合名 = set()

集合名 = set(可迭代对象)

1. 创建具有默认值集合：

集合名 = {1, 2, 3}

集合名 = set(可迭代对象)

1. 添加元素：

集合名.add(元素).

1. 删除元素：

集合名.discard(元素)

"""

**集合set**

**价值1：去重复**

"""

# 1. 创建

set01 = {"悟空", "八戒", "唐三藏"}

list01 = [10, 20, 30, 20, 30, 40]

set02 = set(list01)

print(set02)

# 2. 添加 集合名.add(元素)

set01.add("qtx")

# 3. 删除 集合名.remove(元素)

set01.remove("八戒")

# 4. 遍历

for item in set01:

print(item)

### 运算

1. 交集&：返回共同元素。

s1 = {1, 2, 3}

s2 = {2, 3, 4}

s3 = s1 & s2 # {2, 3}

1. 并集：返回不重复元素

s1 = {1, 2, 3}

s2 = {2, 3, 4}

s3 = s1 | s2 # {1, 2, 3, 4}

1. 补集-：返回只属于其中之一的元素

s1 = {1, 2, 3}

s2 = {2, 3, 4}

s1 - s2 # {1} 属于s1但不属于s2

补集^：返回不同的的元素

s1 = {1, 2, 3}

s2 = {2, 3, 4}

s3 = s1 ^ s2 # {1, 4} 等同于(s1-s2 | s2-s1)

1. 子集<：判断一个集合的所有元素是否完全在另一个集合中
2. 超集>：判断一个集合是否具有另一个集合的所有元素

s1 = {1, 2, 3}

s2 = {2, 3}

s2 < s1 # True

s1 > s2 # True

1. 相同或不同== !=：判断集合中的所有元素是否和另一个集合相同。

s1 = {1, 2, 3}

s2 = {3, 2, 1}

s1 == s2 # True

s1 != s2 # False

子集或相同,超集或相同 <= >=

"""

**集合set**

**价值2：数学运算**

"""

s1 = {1, 2, 3}

s2 = {2, 3, 4}

# 1. 交集&：返回共同元素。

s3 = s1 & s2 # {2, 3}

# 2. 并集：返回不重复元素

s4 = s1 | s2 # {1, 2, 3, 4}

# 3. 补集-：返回只属于其中之一的元素

s5 = s1 - s2 # {1} 属于s1但不属于s2

s6 = s2 - s1 # {4}

# 补集^：返回不同的的元素

s7 = s1 ^ s2 # {1, 4} 等同于(s1-s2 | s2-s1)

# 4. 子集 <：判断一个集合的所有元素是否完全在另一个集合中

# 5.超集 >：判断一个集合是否具有另一个集合的所有元素

s8 = {2, 3}

# s8 子集 s1 超集

print(s8 < s1) # True

print(s1 > s8) # True

### 集合推导式

1. 定义：

使用简易方法，将可迭代对象转换为集合。

1. 语法:

{表达式 for 变量 in 可迭代对象}

{表达式 for 变量 in 可迭代对象 if 条件}

## 容器总结

总结容器：统一管理数据

字符串str:储存字符编码值,不可变,序列

列表list:储存变量,可变,序列

元组tuple:储存变量,不可变,序列

字典dict:储存键值对,可变,散列

键不能重复且不可变

集合set:储存键,可变,散列

不可变：数据在内存中本质都是不可变,采用按需分配的存储机制

可变：具有扩容能力,采用预留空间的存储机制

序列：相邻有序,定位灵活(索引、切片)

散列：分散无序,定位迅速(键)

列表:存储单一维度的数据,例如:治愈人数列表,地区列表

字典：存储多个维度的数据,例如:学生信息,疫情列表

# 1. 创建

dict01 = {"a": "A", "b": "B"}

list01 = [10, 20, 30]

list02 = list(dict01) # ['a', 'b']

list03 = list(dict01.values()) # ['A', 'B']

list04 = list(dict01.items()) # [('a', 'A'), ('b', 'B')]

# 列表转换为字典的格式要求：列表中元素必须能够一分二

list05 = [('a', 'A'), ('b', 'B')]

dict02 = dict(list05) # {'a': 'A', 'b': 'B'}

# 2.添加

list01.append(40)

list01.insert(1, 50)

dict01["c"] = "C"

# 3. 定位 容器名称[整数] 容器名称[开始:结束:间隔]

print(list01[0])

list01[-1] = 100

print(list01[:2])

# 遍历右侧,依次存入左侧

list01[-2:] = "悟空"

# import random

# # 包含开始结束

# random.randint(1,10)

# 删除

list01.remove(20)

del list01[0]

print(list01)

# 遍历

# 从头到尾读取

for item in list01:

print(item)

# 根据索引定位

for i in range(len(list01)):

print(list01[i])

# 键值对

for k,v in dict01.items():

print(k)

print(v)

# 键

for k in dict01:

print(k)

# 值

for v in dict01.values():

print(v)

# 函数 function

## pycharm快捷键

Ctrl + P 参数信息（在方法中调用参数）

Ctrl + Q 快速查看文档

Ctrl + Alt + M 提取方法

## 定义

1. 用于封装**一个**特定的功能，表示一个功能或者行为。
2. 函数是可以重复执行的语句块, 可以重复调用。

## 作用

提高代码的可重用性和可维护性（代码层次结构更清晰）。

"""

**函数 - 功能**

**价值1：减少代码的重复**

**主体思想：做+用分开**

**将共性行为（动词）提取到函数中，将数据（具体变抽象）的变化作为参数**

**将制作与使用分开（制作改一次，使用多次同步修改）**

**当做法变了，不影响用法**

制作(1次)

def 函数名称():

函数体

使用（多次）

函数名称()

"""

# 重复的代码就是万恶之源

# 做 + 用

# print("摆拳")

# print("勾拳")

# print("侧踹")

# print("正蹬")

# print("直拳")

# # ........

# 做 + 用

# print("摆拳")

# print("勾拳")

# print("侧踹")

# print("正蹬")

# print("直拳")

# 做

def attack():

print("摆拳")

print("勾拳")

print("侧踹")

print("正蹬")

print("直拳")

print("发大招")

# 用

attack()

attack()

## 定义函数

1. 语法：

def 函数名(形式参数):

  函数体

1. 说明：

def 关键字：全称是define，意为”定义”。

函数名：对函数体中语句的描述，规则与变量名相同。

形式参数：方法定义者要求调用者提供的信息。

函数体：完成该功能的语句。

1. 函数的第一行语句建议使用文档字符串描述函数的功能与参数。

"""

**函数 - 功能**

**价值2：做与用之前可以通过参数传递信息（并且数据灵活）**

**参数：调用函数 给 制作函数 传递的信息**

练习：exercise01,02

"""

# 做

def single\_attack():

print("摆拳")

print("勾拳")

print("侧踹")

print("正蹬")

print("直拳")

print("发大招")

# 形式参数

def multiple\_attacks(count):

# count = 10

for \_\_ in range(count):

print("摆拳")

print("勾拳")

print("侧踹")

print("正蹬")

print("直拳")

print("发大招")

# 用

# 实际参数

multiple\_attacks(10)

multiple\_attacks(3)

做给用传递传递信息，靠参数传递，

参数意义：数据不同，行为相同，让一个行为适用各个不同的数据。

参数优势：灵活，将数据变得灵活

## 调用函数

1. 语法：函数名(实际参数)
2. 说明：根据形参传递内容。

"""

**函数 - 功能**

**价值3：函数调用推导过程，执行原理**

**程序自生而下运行**

**创建函数的代码要先加载到内存中(不执行)**

**再去调用函数（调用函数必须放在下面，只有创建的制作函数加载后才可调用）**

"""

# --------------创建(定义)函数--------------内部创建函数可改变顺序

注：不是函数嵌套，只是函数内部调用其他函数

def single\_attack():

print("摆拳") # 3

print("勾拳")

print("侧踹")

print("正蹬")

print("直拳")

print("发大招")



def multiple\_attacks(count):



# count = 10

for \_\_ in range(count): # 2

single\_attack()

# --------------调用(使用)函数-------------------

multiple\_attacks(2) # 1

multiple\_attacks(5) 写法1

num = 5

multiple\_attacks(num) 写法2

做函数要设身处地为使用者着想，考虑函数的适用性广，通用性强

## 返回值

1. 定义：

方法定义者告诉调用者的结果。

1. 语法：

return 数据

1. 说明：

return后没有语句，相当于返回 None。

函数体没有return，相当于返回None。

手机屏幕的截图

描述已自动生成

"""

**函数 -- 返回值 语法**

**制作函数 给 使用函数 传递的信息**

**def 函数名():**

**函数体**

**return 数据 # 返回 1个结果**

**可通过列表、元组（，）等容器包装多个结果**

**变量名 = 函数名()**

"""

# 做

def func01():

print("func01执行喽")

return 10 # return 数据 返回 结果

# 用

result = func01()

print("函数的返回值是: %d" % result)

# 如果函数没有return,相当于return None

# 如果函数return后面没有数据,也相当于return None

#return直接退出函数，无论几层循环，全部退出

（如果有if return, else不需要写）

#break，退出一层循环体

def func02():

print("func01执行喽")

# return None

return

result = func02()

print(result)

"""

**函数 -- 返回值 应用**

**设计理念：崇尚小而精,拒绝大而全**

**练习：exercise03~09**

"""

# # 1. 获取数据

# usd = input("请输入美元：")

# # 2. 逻辑处理

# rmb = float(usd) \* 7.1465

# # 3. 显示结果

# print(usd + "美元 =" + str(rmb) + "人民币")

# 将一个功能从头到尾实现,分割为做法 + 用法

# 获取数据 --> 参数

# 逻辑处理 --> 函数体

# 显示结果 --> 返回值

def usd\_to\_rmb(usd): #1

# 2. 逻辑处理

rmb = float(usd) \* 7.1465 #2

return rmb #3

rmb = usd\_to\_rmb("20")

print(rmb)

## 可变／不可变类型在传参时的区别

1. 不可变类型参数有:

**数值型(整数，浮点数)**

**布尔值bool**

**None 空值**

**字符串str**

**元组tuple**

1. 可变类型参数有:

**列表 list**

**字典 dict**

**集合 set**

1. 传参说明：

不可变类型的数据传参时，函数内部不会改变原数据的值。

因不可变数据不能修改 eg:P1 = “爱国主义” 字符串，

item = P1[0] 只可读 P1[0]=“不爱”不可改，语法错误

可变类型的数据传参时，函数内部可以改变原数据。

有能力改，但不一定做

手机屏幕截图

描述已自动生成

# 1将函数的代码存储到代码区中（只是存储文字，不执行）

def func01(p1, p2): p1得到list01的地址，p2得到list02的地址

p1[0] = 100 改变的是栈帧外元素，通过栈帧变量用了定位技术索引等

p2 = 200 改变的是栈帧中的变量P2，重新赋值(无论传入的参数是 可变对象还是不可变对象)

list01 = [10]

list02 = [20]

# 2调用函数在内存中开辟一块空间(栈帧) 栈帧：类似电影画面，转瞬即逝

# 栈帧用于存储在函数内创建的变量（局部变量）

func01(list01, list02)

# 3函数执行后栈帧释放

print(list01[0]) # 100

print(list02[0]) # ?[20]

"""

**函数内存分布（必须按下述步骤执行）**

**1. 传入可变对象（列表、字典、集合） 可变对象**

**2. 修改可变对象（定位：索引、切片、键） 修改方式**

**3. 函数外可以得到结果(不用return) 打印传入对象**

练习:exercise10

对比

1.传入可变对象

2.修改局部变量

3.return局部变量

"""

## 函数参数

### 实参传递方式argument

#### 位置传参

定义：实参与形参的位置依次对应。

##### 序列传参

定义：实参用\*将序列拆解后与形参的位置依次对应。

#### 关键字传参

定义：实参根据形参的名字进行对应。

##### 字典关键字传参

1. 定义：实参用\*\*将字典拆解后与形参的名字进行对应。
2. 作用：配合形参的缺省参数，可以使调用者随意传参。

"""

**函数参数**

**实际参数 对应**

**1. 位置实参:实参按照位置与形参进行一一对应**

**3.序列实参：使用\*号拆分序列,将元素与形参对应**

**2. 关键字实参：实参按照名称与形参进行对应**

**4.字典实参：使用\*\*号拆分字典,将键值对与形参对应**

"""

全部参数传递时，用位置实参；部分参数传递时，用关键字实参。

中间跳过部分参数时，先按顺序，后按名字传递。（1，P3=10）

def func01(p1, p2, p3):

"""

我是函数

:param p1: 我的第一个参数

:param p2:

:param p3:

:return:

"""

print(p1)

print(p2)

print(p3)

# ctrl + p 查看参数 （调用与定义函数距离可能很远）

# ctrl + q 查看文档

func01(1, 2, 3)  **位置实参，按顺序**

# TypeError: func01() missing 1 required positional argument: 'p3'

# func01(1, 2) 位置实参，传递少了

# TypeError: func01() takes 3 positional arguments but 4 were given

# func01(1, 2, 3, 4) 位置实参，传递多了

func01(p1=1, p2=2, p3=3) **关键字实参：按形参名称对应**

list01 = [1, 2, 3]

str02 = "123"

tuple03 = (1, 2, 3)

func01(\*list01) # 拆  **序列实参 按顺序对应**

dict04 = {"p1": 1, "p2": 2, "p3": 3}可改变顺序{"p1": 1, "p3": 3, "p2": 2}

func01(\*\*dict04) **字典实参 按关键字名称对应**

### 形参定义方式parameter

#### 缺省形参(默认形参)

1. 语法：

def 函数名(形参名1=默认实参1, 形参名2=默认实参2, ...):

函数体

1. 说明：

缺省参数必须自右至左依次存在，如果一个参数有缺省参数，则其右侧的所有参数都必须有缺省参数。

缺省参数可以有0个或多个，甚至全部都有缺省参数。

#### 位置形参

语法：

def 函数名(形参名1, 形参名2, ...):

函数体

#### 命名关键字形参

1. 语法：

def 函数名(\*args, 命名关键字形参1, 命名关键字形参2, ...):  
  函数体

def 函数名(\*, 命名关键字形参1, 命名关键字形参2, ...):

函数体

1. 作用：

强制实参使用关键字传参

#### 不定长形参

##### 星号元组形参

1. 语法：

def 函数名(\*元组形参名):

  函数体

1. 作用：

可以将多个位置实参合并为一个元组

1. 说明：

一般命名为'args'

形参列表中最多只能有一个

##### 双星号字典形参

1. 语法：

def 函数名(\*\*字典形参名):

函数体

1. 作用：

可以将多个关键字实参合并为一个字典

1. 说明:

一般命名为'kwargs'

形参列表中最多只能有一个

#### 参数自左至右的顺序

位置形参 --> 星号元组形参 --> 命名关键字形参 --> 双星号字典形参

"""

函数参数

形式参数:约束实参

默认形参：实参可选（可传位置实参、关键字实参、混合实参）

位置形参：实参必填（可传位置实参、关键字实参、混合实参）

命名关键字形参：实参必须是关键字实参

不定长形参

星号元组形参：将位置实参合并为元组

位置实参数量无限

双星号字典形参：将关键字实参合并为字典

关键字实参数量无限

"""

# 1. 默认形参：实参可选

# 注意：默认形参必须从右到左依次存在（p1, p2, p3 = 1）

def func01(p1= "", p2 = 0, p3 = 1): 默认形参

print(p1)

print(p2)

print(p3)

# 全部传递

func01(1, 2，3) #参数数量不限1/2/3 可传位置实参

# 需要指定某一个形参（不能单独使用，需要配合默认形参）

func01(p2=2) #参数数量不限1/2/3 可传关键字实参

func01(1, p3=3) #参数数量不限1/2/3 可混合传位置实参+关键字实参

# 2. 位置形参：实参必填

def func01(p1, p2, p3): 位置形参

print(p1)

print(p2)

print(p3)

func01(1, 2, 3) #参数数量必须为3 可传相同数量位置实参

func01(p1=1, p2=2, p3=3) 参数数量必须为3 可传数量相同关键字实参

func01(1, 2, p3=3)参数数量必须为3，且关键字实参最右 可传混合实参

# 3. 星号元组形参：将位置实参合并为元组

# 注意1：形参中只有一个

# 注意2：只支持位置实参

def func02(\*args):

print(args)

func02()

func02(1, 2, 3)

# func02(a=1,b=2)

# 4. p1, p2 是命名关键字形参:

# 限制实参必须是关键字实参

def func03(\*args, p1, p2): p1, p2必须传参

print(args)

func03(1, 2, 3, 4, 5, p1=1, p2=2)

# p1 是必填的信息 主

# p2 是可选的信息 次

def func04(p1, \*, p2=0): \*不需传参，p2可传参也可不传参，如传必关键字

print(p1)

print(p2)

func04(1, p2=2)

# print(10,20,30,end = " ")主+次（结束）

print(10, 20, 30, sep="------")主+次（连接）

print(10, 20, 30, sep="------", end=" ")主+次（结束+连接）

# 如果木有命名关键字形参的技术

# 下列代码可读性非常差的写法,就不能避免了.

# print(10,20,30,"------"," ")

# 5. 双星号字典形参：关键字实参数量无限

def func04(\*\*kwargs):

print(kwargs)

func04()

func04(a=1, b=2, c=3)

exercise

def func01(\*args, \*\*kwargs): # 实参数量无限

print(args)

print(kwargs)

func01() 可不给

func01(1, 2, a=1, b=2) 可给多个

----------------

# p1:位置形参:必填

# p2:位置形参+默认形参:可选

# args:星号元组形参：位置实参数量无限 #有序

# p3:命名关键字形参+默认形参:关键字实参(可选)

# kwargs:双星号字典形参：关键字实参数量无限 #无序

def func02(p1, p2="", \*args, p3=0, \*\*kwargs):

print(p1)

print(p2)

print(args)

print(p3)

print(kwargs)

func02(1,2,3,4,5,p3 = 3,a=1,b=2) 可填多个

func02(1) 可不填的

### 函数参数语法总结与参数应用

"""

**函数参数 - 总结语法**

**实际参数：对应**

1.位置实参：顺序

函数名(数据1,数据2,数据3)

3. 序列实参：拆

函数名(\*序列)

2.关键字实参：名字

函数名(参数名2=数据2)

4. 字典实参：拆

函数名(\*\*字典)

**形式参数：约束**

1. 默认形参：可选

def 函数名(参数名1 = 数据1,参数名2=数据2)

2. 位置形参：必填

def 函数名(参数名1,参数名2,参数名3)

3. 命名关键字形参：必须是关键字实参

def 函数名(\*,参数名1,参数名2)

def 函数名(\*args,参数名1,参数名2)

4. 不定长形参：长度无限

4.1 星号元组形参：合并位置实参

def 函数名(\*args)

4.2 双星号字典形参：合并关键字实参

def 函数名(\*\*kwargs)

"""

def func01(p1=0, p2=1, p3=2):

print(p1)

print(p2)

print(p3)

func01(p1="a", p2="b", p3="c")

def func02(\*,p1,p2):

print(p1)

print(p2)

func02(p1 =1,p2 = 2)

"""

**函数参数 - 应用**

**自学函数常用功能：（字符串、列表、字典、集合）**

**1.crtl + q 查看pycharm文档**

**2.查看官方文档(内置模块最全)：https://docs.python.org/zh-cn/3/**

**3.查看https://www.runoob.com/python3/python3-tutorial.html**

"""

list01 = [432, 545, 6, 7, 6]

print(list01.count(6))

# 扩展(一次追加多个数据)

list01.extend(["a", "b", "c"])

print(list01)

dict02 = {"a": "A"}

dict03 = {"b": "B"}

# 更新(一次增加多个键值对)

dict02.update(dict03)

print(dict02)

str03 = "函数参数"

str03.join str03.split

# 作用域LEGB

1. 作用域：变量起作用的范围。
2. Local局部作用域：函数内部。
3. Enclosing 外部嵌套作用域 ：函数嵌套。
4. Global全局作用域：模块(.py文件)内部。
5. Builtin内置模块作用域：builtins.py文件。（Crtl+鼠标左键能找到创建函数的地方）

"""

作用域

**局部作用域：函数内部有效**

**全局作用域：整个文件有效**

**小范围(一个函数)使用局部变量 （操作的数据在小范围内使用）**

**大范围(多个函数)使用全局变量 （函数内只使用全局变量时不需传参）**

"""

# ------------全局作用域：整个文件有效------------

b = 200 2.全局变量 放在最顶部

# ------------函数------------

def func01():

# 局部作用域：函数内部有效

a = 100 1.局部变量

print(a)

def func02():

# 3.局部可以访问全局（整个文件有效）

print(b)

# ------------调用(入口)------------

func01()

func02()

print(b)

**以上是写程序的基本格式 \*\*\*\*\***

"""

**作用域**

"""

a = 100

def func01():

# 不能修改全局变量

# a = 200 # (创建了新的局部变量)

# 如果局部作用域修改全局变量：必须声明变量

global a

a = 200 # 修改全局

func01()

print(a)

list01 = [10]

def func02():

# 读取全局变量（没有修改全局变量 list01=[20]才是修改全局变量）

# 修改列表第一个元素

list01[0] = 20

func02()

print(list01) # [20]

## 变量名的查找规则

1. 由内到外：L -> E -> G -> B
2. 在访问变量时，先查找本地变量，然后是包裹此函数外部的函数内部的变量，之后是全局变量，最后是内置变量。

## 局部变量

1. 定义在函数内部的变量(形参也是局部变量)
2. 只能在函数内部使用
3. 调用函数时才被创建，函数结束后自动销毁

## 全局变量

1. 定义在函数外部,模块内部的变量。
2. 在整个模块(py文件)范围内访问（但函数内不能将其直接赋值）。

## global 语句

1. 作用：

在函数内部修改全局变量。

在函数内部定义全局变量(全局声明)。

1. 语法：

global 变量1, 变量2, …

1. 说明

在函数内直接为全局变量赋值，视为创建新的局部变量。

不能先声明局部的变量，再用global声明为全局变量。

## nonlocal 语句

1. 作用：

在内层函数修改外层嵌套函数内的变量

1. 语法

nonlocal 变量名1,变量名2, ...

1. 说明

在被嵌套的内函数中进行使用