**python学习**

01. 恢复pycharm的初始设置

Pycharm的配置信息在用户家目录的.PyCharmCE2019.2目录下, CE2019.2是PyCharm的版本号.

若要回复pycharm的初始设置, 按照如下步骤:

(1)关闭正运行的pycharm

(2)在终端执行rm -r ~/.PyCharmCE2019.2(删除配置信息文件目录)

(3)重新启动pycharm

02.第一次启动Pycharm

1. 导入配置信息

2. 选择许可协议(pycharm付费软件),学生和老师免费版本

3. 配置初始界面(选择编辑器的配色方案)

03. 新建/打开一个python项目

项目 –- 解决复杂业务功能的软件

(1)一个项目具有一个独立专属目录.

(2) 打开项目(目录),打开后会在目录下新建一个.idea文件, 用于保存项目的相关信息,如:解释器的版 本项目包含的文件信息等.

### 128解释器版本设置

04. 设置项目使用的解释器版本

File-settings-Project:[项目名]-Ptoject Interpreter

### 129 项目和文件名的命名规则



1. 命名规则

注意:

1. 命名文件名时建议只是用小写字母, 数字和下划线

2. 文件名不能以数字开头

课程中命名hm\_xx\_知识点 方式命名,其中xx为序号

### 130 新建项目演练

1. 新建项目方式:

(1)欢迎界面

(2)File/New Project

新建01\_python基础项目,新建hm\_01\_hello.py文件.

编写print(“Hello World”)代码

### 131 编辑区域和控制台的字体显示

控制台字体 settings-Editor-Color & Fonts-Color Scheme-Console Font-Primary font: DialogInput, 修改size

编辑区字体 settings-Editor-Color & Fonts-Font-Font: DialogInput, 修改size

### 132 pycharm的升级与启动

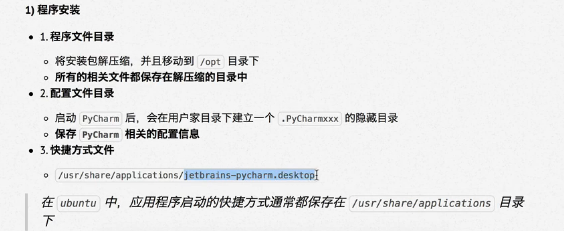
1. 教育版 https://www.jetbrains.com/pycharm-edu/download/#section=linux

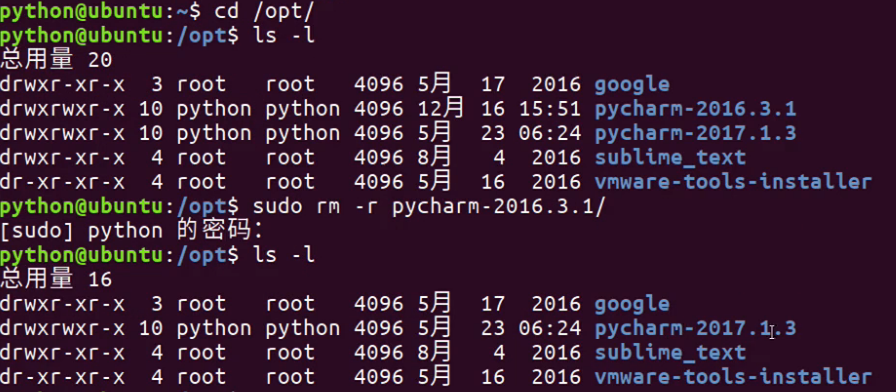
2. 专业版 [https://www.jetbrains.com/pycharm/download/#section=linux](https://www.jetbrains.com/pycharm/download/" \l "section=linux)

3. tar解压及启动



进入家目录cd, 删除配置文件

卸载其他版本的pycharm ,删除原来的版本(/opt/文件下)及其配置文件(在家目录下的隐藏文件中 ls -la)

 2. edu版安装

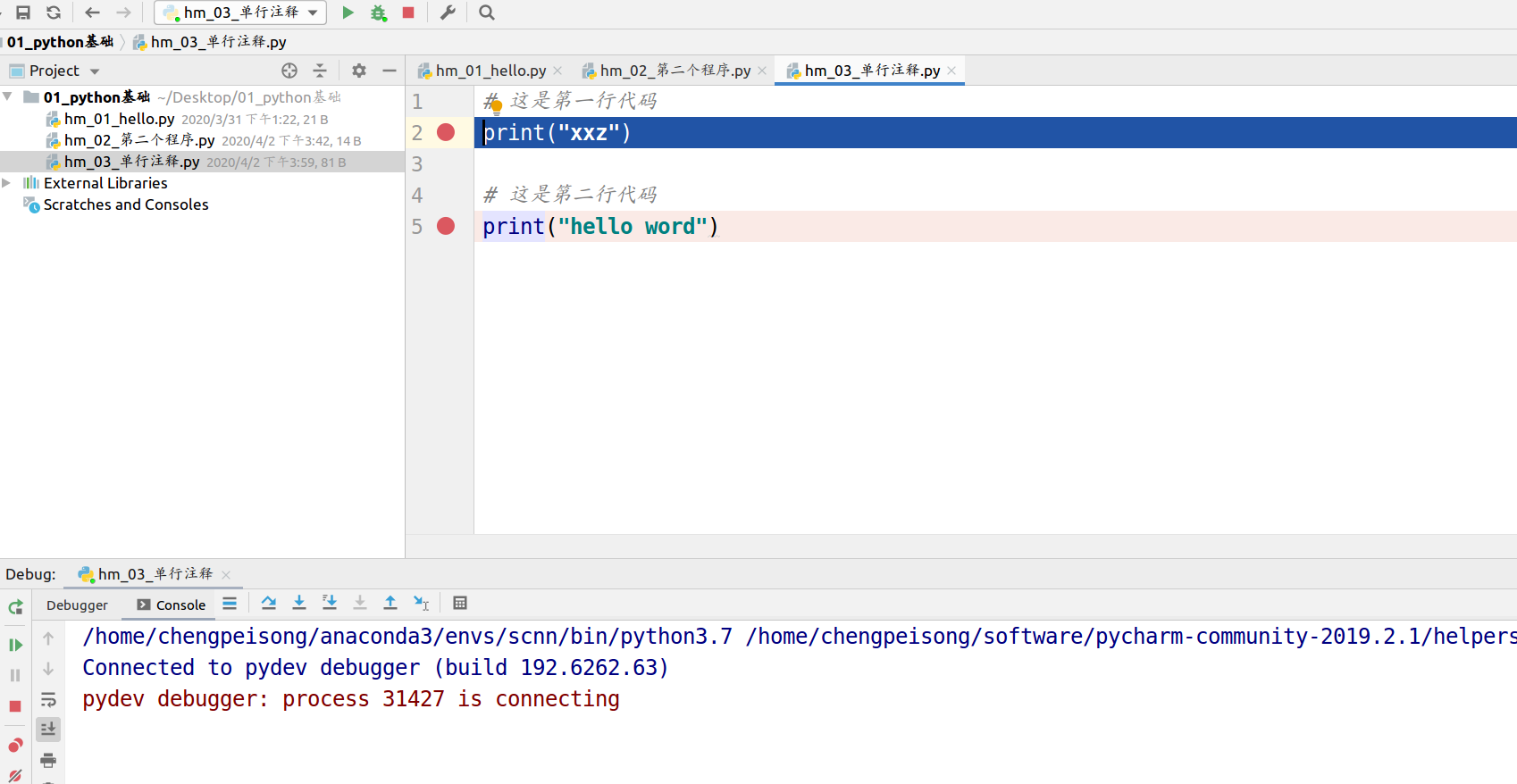
解压 tar -zxvf pycharm-edu-3.5.1.tar.gz

将edu的目录移动到opt文件下 sudo mv -r pycharm-edu-3.5.1 /opt/

python知识点学习

1. 单步执行

在代码前点击,出现代码前出现红色的点号,点击debug绿色小虫子(shift+F9), 点击控制台中的Console一步一步的执行(F8)

2.单行注释3. 单行注释(代码末尾添加#)

4. 多行注释("""123""")

* 01. 算数运算

+ - \* /

/ 除 如:9/2输出4.5

// 取整除 如:9//2输出4

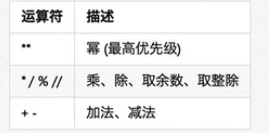
% 取余数 如:9%2输出1

\*\* 幂 又叫次方,乘方 如: 2\*\*3输出8

print("\*\*"\*50) 输出字符串

* 02.算数运算符的优先级

可以用括号改变优先级



147 计算机中的三大件:

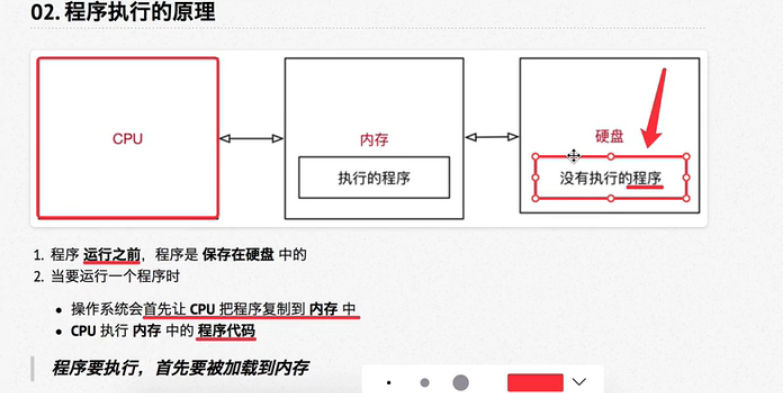
CPU(中央处理器,处理数据/计算)

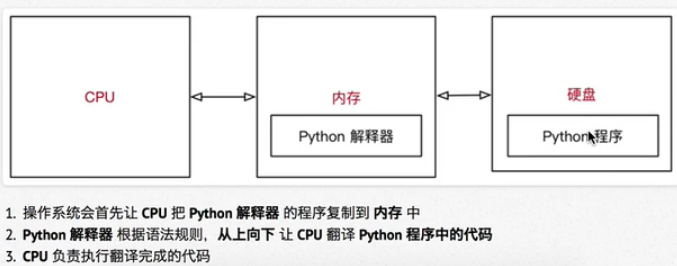
硬盘(永久存储数据,速度慢,空间大)

内存(临时存储,速度快,空间小)

由于内存的运行速度快,所有CPU会将硬盘中的文件复制到内存中,CPU执行内存中的程序代码

03.python执行原理



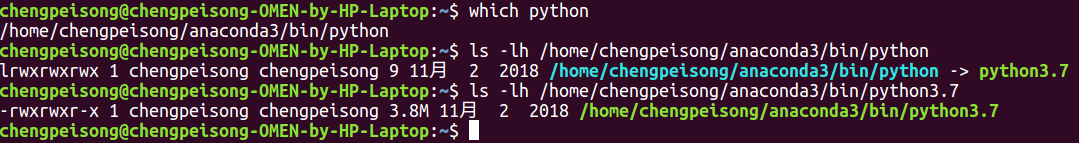


(1)在终端中用Ctrl+d可以退出python程序

(2)which python可以查看python解释器保存在哪个目录下

->软链接

(3)查看python解释器的内存 3.8M ls -lh 目录名

注意:建立软链接的目的是:为方便使用者不用记住解释器的版本

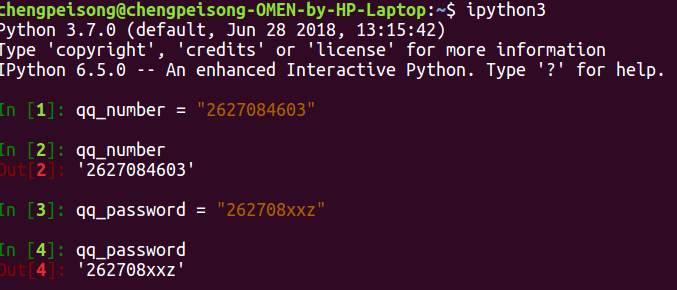
程序是用来处理数据的,变量是来存储数据的.变量是程序内部为了保存数据,开辟的内存空间.

**153.变量的使用**

**01.变量定义**

**变量名 = 值**

**(1)打开ipython, 在终端中输入ipython3, 退出输入exit或者ctrl+d询问是否退出**



(2) pycharm中演练

qq\_number = **"2627084603"**

qq\_password = **"262708xxz123"**

print(qq\_number)

print(qq\_password)

(3)超市买苹果

可以用其他变量的计算结果来定义变量

变量定义后,后续可以直接使用

# 定义苹果的单价 8.5/斤

price = 8.5

# 2. 挑选苹果(7.5斤)

weight = 7.5

# 3. 计算付款金额

money = weight \* price

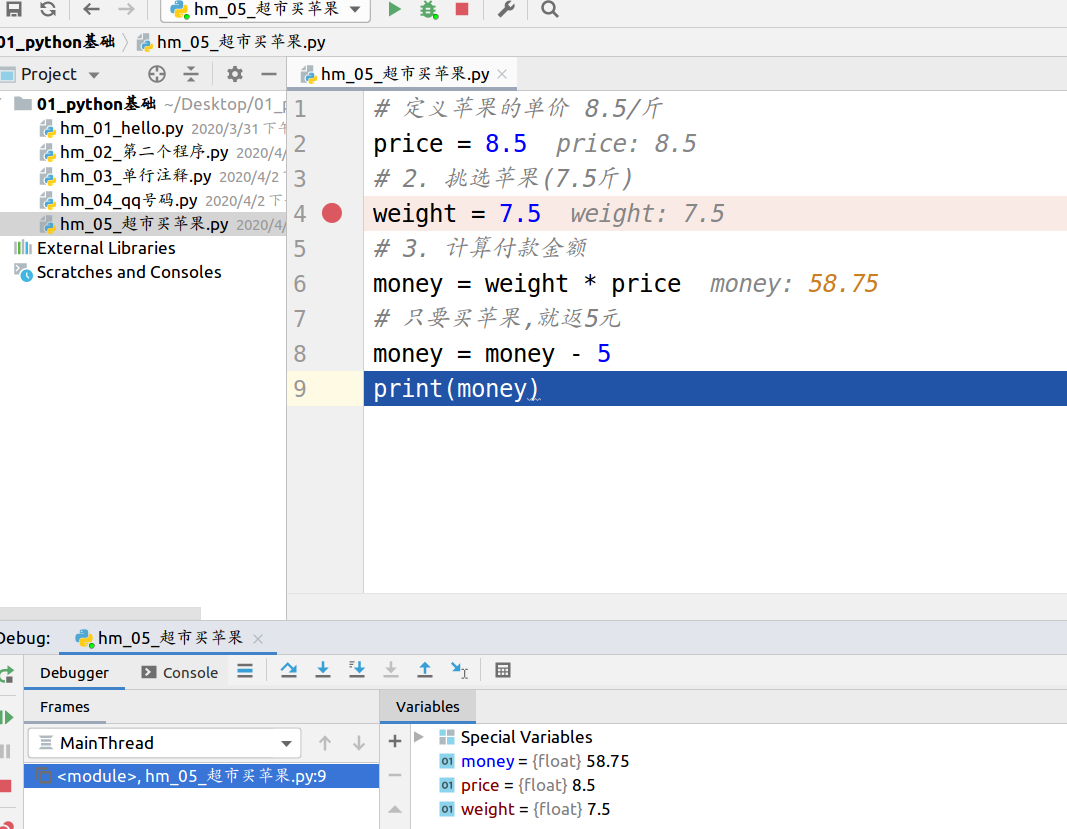
# 买苹果就返5元

money = money - 5

print(money)

pycharm单步执行查看变量

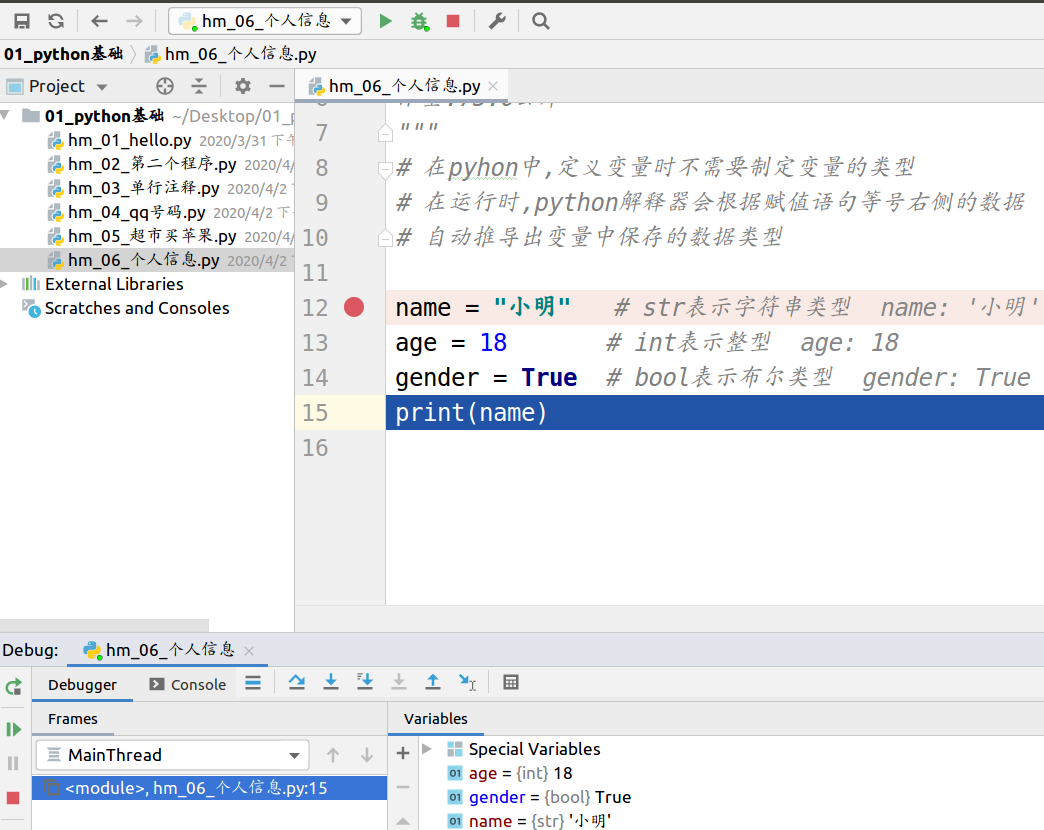
红点-debug-debugger(查看变量)-Console(单步调试,最终结果,变量释放)



变量名只有在第一次出现时,才定义变量,再次使用不是重新定义,变量中存储的值始终是变化的

**02. 变量类型**

* int(整型)
* float(浮点型)
* bool(布尔型)
* str(字符串型)



代码调试

**shift+F9 调试程序,停在断点处**

**F8：step over 单步**

**F7：step into 进入**(配合F8使用。单步调试F8时，如果某行调用其他模块的函数，在此行F7，可以进入函数内部，如果是F8则不会进入函数内容，直接单步到下一行)

##### Alt+shift+F7：step into mycode

个人理解F8和F7的综合。1、没遇到函数，和F8一样；2、遇到函数会自动进入函数内部，和F8时按F7类似的

##### shift+F8：跳出

调试过程中，F7进入函数内后，shift+F8跳出函数，会回到进入前调用函数的代码。不是函数地方shift+F8跳出，怎么用没太明白，但最终会执行到结束。

**F9：resume program** (按翻译是重启程序 ，实际是下个断点，当打多个断点时，F9会到下一个断点,继续执行, 释放内存, 节约内存)

**Ctrl+F2 停止调试**

**161. python中的变量类型**

1. 数字型和非数字型

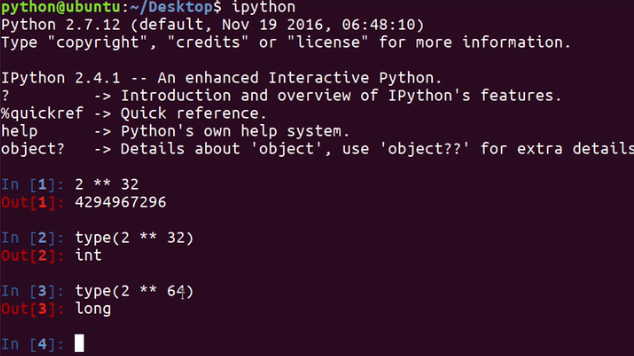
(1)数字型

* 整型 (int)
* 浮点型(float)
* 布尔型(True(非零数) False(0), 非零即真)
* 复数型(complex,主要用于科学计算,例如:平面场问题,波动问题,电感电容问题等)

(2)非数字型

* 字符串(str)
* 列 表(list)
* 元 组(tuple)
* 字 典(dict)

python2.x中整数可以有int(整型)和long(长整型)



2. 查看变量的类型--------type(x)

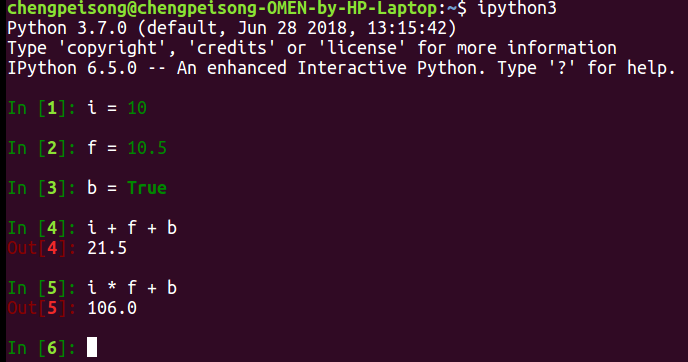
print(type(name)) ----→ <class 'str'>

**164. 变量的计算**

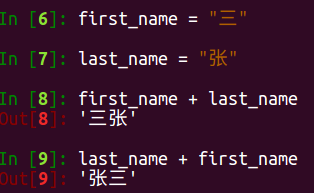
1. 不同类型变量之间的计算

(1)数字型变量之间直接计算

如果变量是bool类型,在计算时, 真为1, 假为0.



(2) 字符串变量之间使用 + 拼接字符串



(3) 字符串变量可以和整数使用 \* 重复拼接相同的字符串



(4)python中,数字型变量和字符串之间不能进行其他计算

**166. 变量的输入输出**

输入就是用代码获取用户通过键盘输入的信息.

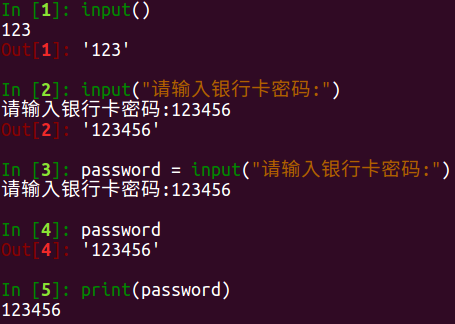
用input获取用户在键盘上的输入信息, 输的是字符串类型的信息.

**(1) 关于函数**

目前学到的函数 **直接使用,不需要关心内部的细节**

print(x) 将x输出到控制台

type(x) 查看x的变量类型

****

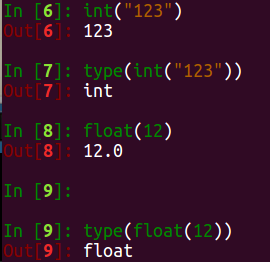
**(2) input函数实现键盘输入**

* 在python中,可以使用input函数从键盘等待用户的输入
* 用户输入的任何内容python都会认为是一个字符串
* **语法如下:**

变量名 = input(“提示信息:”)

**(3) 类型转换函数**

* int(x) 将x转换为整数
* float(x) 将x转换为浮点数



**(4)变量的格式化输出 %**

苹果单价9.00 元/斤,购买了5.00斤,需要支付45.00元.

* 在python中可以使用print函数将信息输出到控控制台
* 若想输出信息时, 一起输出数据, 就需要使用到格式化操作符
* % 被称为格式化操作符,专门用于处理字符串中的格式

|  |  |
| --- | --- |
| 格式化字符 | 含义 |
| %s | 字符串,如:name = “小明”  print(“我的名字是%s” % name) |
| %d | 有符号十进制, %06d表示输出的整数显示位数, 不足6位的地方前面用0补全, 如:number = 100 print(“我的学号是%06d” % number) -----> 我的学号是000100 |
| %f | 浮点数, %.02f表示小数点后只显示两位  如: price = 8.5  weight = 7.5  money = price \* weight  print(“苹果的单价是%.2f元/斤, 购买了%.2f斤, 需要支付%.2f元”%(price, weight, money)) |
| %% | 输出%  如: scale = 0.25  print(**"数据比例是%.2f%%"** % (scale \* 100))----> **数据比例是**25.00% |

print(“格式化字符串”% 变量1)

print(“格式化字符串”%(变量1, 变量2…))

**179 变量的命名**

**01. 标识符和关键字**

**1.1 标识符**

* 标识符就是程序员定义的**变量名, 函数名**

名字需要**有见名知义**的效果

* 标识符可以由**字母,数字和下划线**组成
* 不能以数字开头
* 不能与关键字重名

**1.2 关键字以及查看关键字列表**

关键字就是在python内部已经使用的标识符

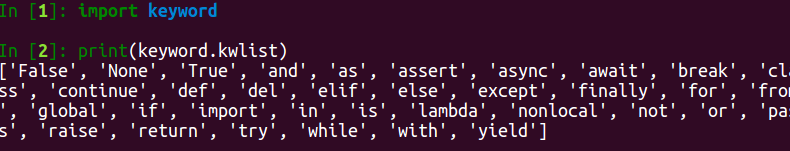
关键字具有特殊的功能和含义

**开发者不允许定义和关键字相同的名字和标识符**

查看python中的关键字

import keyword

print(keyword.kwlist)



**02. 变量的命名规则**

**python中是严格区分大小写的**

1. 定义变量时,为了保证代码格式, = 的左右应该各保留一个空格

**小写字母和下划线组合命名法**

2. 在python中, 如果变量名需要两个或多个单词组成时,可以按照以下命名规则命名:

a. 每个单词都使用小写字母

b. 单词与单词之间使用\_(下划线)连接

c. 例如: first\_name, qq\_number

**驼峰命名法**

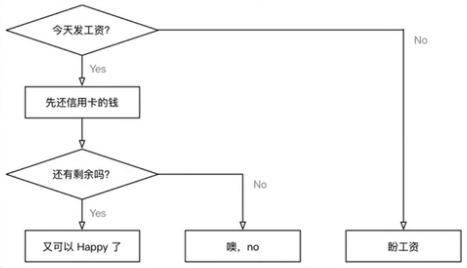
当变量名是由二个或多个单词组成时, 还可以利用驼峰命名法来命名.

* 小驼峰命名法:
* 第一个单词以小写字母开始,后续单词的首字母大写
* 例如: firstName, lastName
* 大驼峰命名法:
  + 每个单词的首字母采用大写字母
  + 例如: FirstName, LastName, CamelCase

**182. 判断(if)语句(又叫分支语句)**

**01. 开发中的应用场景**

网吧 安检 程序中的判断



if 今天发工资:

先还信用卡的钱

if 还有剩余:

又可以happy了,哈哈

else:

奥,NO

else:

盼工资

**02.if语句**

**2.1 判断语句**

if 要判断的条件:

条件成立时,要做的事情

……

**注意:**

* 代码的缩进为一个**Tab**键,或者4个空格-----**建议使用空格**
* 在python开发中,Tab键和空格不能混用!

**2.2 代码演练----年龄判断**

# 1.定义一个整型变量记录年龄

age = 18

# 2.判断年龄是否满了18岁

**if** age >= 18:

# 3.如果满了18岁,可以进网吧嗨皮

print(**"您可以成年,可以进入网吧嗨皮"**)

if语句以及缩进部分看成完整的代码块

**运算符**

01. 比较运算符

== 相等

!= 不等于 (python2.x中的<>和!=都可以表示不等于)

> 大于

< 小于

>= 大于等于

<= 小于等于

**2.3 else处理条件不满足的情况(配合if一起使用)**

格式:

if 要判断的条件:

条件成立时,要做的事情

…….

else:

条件不成立时,要做的事情

…….

**2.4 判断年龄的改进**

**ctrl + / 备注**

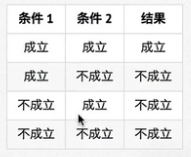
**192.逻辑运算符**

* 程序开发中,多个条件都满足,才能够执行后续程序,这时要用到逻辑运算符.
* 逻辑运算符可以把多个条件按照逻辑进行拼接,变成更复杂的条件.
* python中的逻辑运算符包括**与(and), 或(or), 非(not)**三种

**3.1 and**

条件1 and 条件2

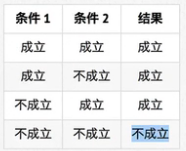
* 两个条件同时满足,返回True
* 只要由一个不满足, 就返回False



**3.2 or**

条件1 or 条件2

* 只要有一个条件满足, 就返回True
* 两个条件都不满足, 返回False



**3.3 not**

not 条件

非/不是



**194. 逻辑运算演练-----考试成绩**

**or 演练**

**196. elif语法**

**1. elif**

再增加一些条件,需要执行的代码也不同时,就可以用elif

if 条件1:

条件1满足执行的代码

elif 条件2:

条件2满足执行的代码

…….

else:

以上都不满足时,执行的代码

注意:

1. elif和else都必须和if一起使用,而不单独使用

2. 可以将if, elif和else以及各自的缩进的代码.看成一个完整的代码块.

# 判断要用==

代码:

# 定义holiday\_name字符串变量记录节日名称

holiday\_name = **"生日"**

# 如果是情人节,应该买玫瑰/看电影

# 判断要用两个=

**if** holiday\_name == **"情人节"**:

print(**"应该买玫瑰"**)

print(**"应该看电影"**)

# 如果是平安夜,应该买苹果/吃大餐

**elif** holiday\_name == **"平安夜"**:

print(**"应该买苹果"**)

print(**"应该吃大餐"**)

# 如果是生日.应该买蛋糕

**elif** holiday\_name == **"生日"**:

print(**"应该买蛋糕"**)

# 其他的日子每天都是节日啊......

**else**:

print(**"每天都是节日啊!"**)

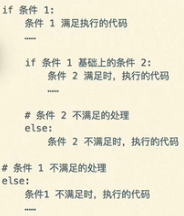
elif的应用场景:同时判断多个条件,所有的条件都是平级的.

2.if嵌套

应用场景: 在之前条件满足的前提下, 再增加额外的判断

if的嵌套的语法格式,除了缩进之外和之前的没有区别

语法格式如下:

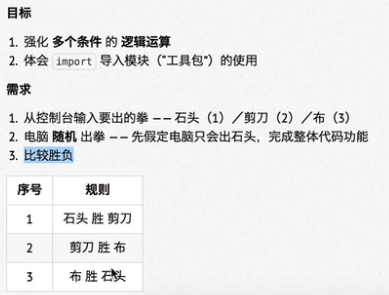


注意对齐格式

Tab键向右缩进

Shift+Tab键向左缩进

203. 石头剪刀布游戏演练



在大条件前添加一个小括号, 每个小条件回车, 自动换行

# 从控制台输入要出的拳----石头(1)/剪刀(2)/布(3)

player = int(input(**"请输入您要出的拳--石头(1)/剪刀(2)/布(3):"**))

# 电脑出拳

computer = 1

print(**"玩家选择的拳头是%d, 电脑选择的拳头是%d"** %(player, computer))

# 比较胜负

# 1.石头 胜 剪刀

# 2.剪刀 胜 布

# 3.布 胜 石头

**if** ((player == 1 **and** computer == 2)

**or** (player == 2 **and** computer == 3)

**or** (player == 3 **and** computer == 1)):

print(**"玩家胜利,电脑弱爆了"**)

# 平局

**elif** player == computer:

print(**"真是心有灵犀,再来一局"**)

# 其他情况电脑胜利

**else**:

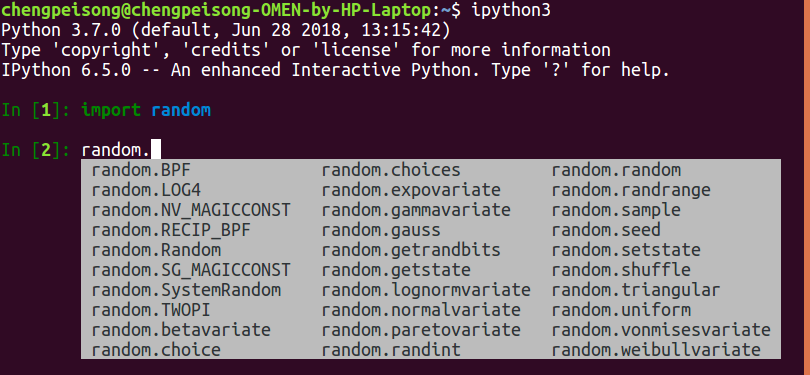
print(**"不服气,我们决战到天亮!"**)

**2. 随机数的处理**

在python中,要使用随机数,首先需要导入随机数的模块---”工具包”

import random

导入模块后,直接在模块名称后面敲一个.然后按Tab键,会提示该模块中包含的所有函数



random.randint(a,b),返回[a,b]之间的整数,包含a和b

type(random.randint(10,20))----->int

random.randint(10,10) # 返回数字10

random.randint(1,10) # 返回1-10之间的任意一个整数

random.randint(10,1) # 报错,下限必须小于上限

209 循环

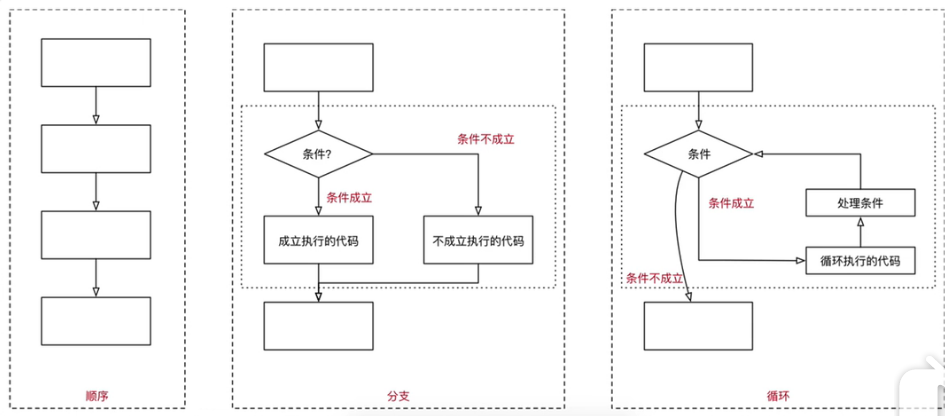
程序的三大流程

* while循环的基本使用
* break和continue
* while循环嵌套

01. 程序的三大流程

* **三种流程方式**

1. **顺序----从上向下,顺序执行代码**
2. **分支----根据条件判断,决定执行代码的分支(if-else, if-elif-……-else)**
3. **循环----让特定代码重复执行(for, while)**

**211. while循环**

作用: 让指定的代码重复执行

应用场景: 让执行的代码按照指定的次数重复执行

基本语法:

初始条件设置----通常是重复执行的计数器

i = 1

while 条件(判断计数器是否达到目标次数):

条件满足时,做的事情1

条件满足时,做的事情2

……(省略)……

处理条件(计数器 + 1) i=i+1

**注意:**

while语句以及缩进部分是一个**完整的代码块**

**第一个while循环**

**需求:**

**打印5遍 Hello Python**

**代码:**

# 1.定义一个整数变量,记录循环次数

i = 1

# 2.开始循环

**while** i <= 5:

# <1>希望在执行内部执行的代码

print(**"Hello Python"**)

# <2>处理计算器(i = i + 1)

i += 1

# 3. 观察一下,循环结束后,计算器i的数值是多少

# 循环结束后,之前定义的计数器条件的数值是依旧存在的

print(**"循环结束后,i = %d"** % i) #--→6

214 死循环

**死循环**

由于程序员的原因,忘记在循环内部修改循环的判断条件,导致循环继续执行,程序无法终止.如:不加i= i+1,记住加上计数器的变化.

215 赋值运算符

=

+=

-=

\*=(乘)

/=(除)

//=(整除)

%=(取余)

\*\*=(乘方)

2.3 python中的计数方法

程序计数从0开始

2.4 循环计算

在程序开发中,通常会遇到利用循环重复计算的需求(例如:求和计算)

遇到这种需求时,可以:

1.在while上方定义一个变量,用于存放最终计算结果

2.在循环体内部,每次循环都用最新的计算结果,更新之前定义的变量

需求1:

计算1-100的累加和

# 计算0-100之间所有数字的累计求和结果

# 0.定义一个变量，记录最终结果

result = 0

# 1.定义一个整数变量,记录循环次数,计数器

i = 0

# 2.开始循环

**while** i <= 100:

# print(i)

# 每一次循环，都让result这个变量和i计数器进行相加

result += i

i += 1

print(**"1~100的累加和是：%d"** % result)--->5050

**需求2**：

计算0~100之间所有偶数的累加和的结果

**偶数(x%2==0)**

**奇数(x%2!=0)**

**# 计算0～100之间所有偶数的累加和的结果**

# 开发步骤：

# 1.编写循环 确认 要计算的数字

# 2.添加 结果 变量，在循环体内部，处理计算结果

result = 0

i = 0

**while** i <= 100:

# 1.判断偶数 i % 2 == 0:

# 判断奇数 i % 2 != 0:

**if** i % 2 == 0:

# print(i)

# 2.当i是偶数时，才进行累加

result += i

# 3.计数器累加,和while是配套使用的

i += 1

print(**"0～100之间的偶数和为:%d"** % result) #--->2550

# print("0～100之间的奇数和为:%d" % result) #--->2500

221.break和continue语句

03.break和continue

* break和continue是专门在循环体中使用的关键字
* break某一条件满足时，退出所有循环，不再执行后续重复的代码
* continue某一条件满足时，不执行后续重复的代码,退出当前循环
* break和continue只针对当前所在循环有效

**(1)break演练:**

i = 0

**while** i < 10:

# break 某一条件满足时，退出所有循环，不执行后续的重复代码

# 4-9不会执行,直接跳出while循环

**if** i == 3:

# 直接跳出循环,执行打印over

**break**

print(i)

i += 1

print(**"over"**)

打印出0,1,2 over(一行行)

**(2)continue演练:**

**i = 0**

**while** i < 10:

# continue当某一条件满足时，不执行continue后续重复的代码,

# 退出当前循环,回到while

**if** i == 3:

# 注意：在循环中，使用continue这个关键字前，需要确认循环的计数是否修改

# 不加i += 1,会造成死循环

i += 1

**continue**

print(i)

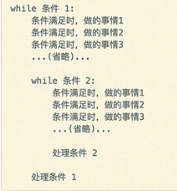
i += 1

打印出0,1,2,4,5,6,7,8,9

225 while循环嵌套

4.1 循环嵌套

* while嵌套就是：while里面还有while



**演练1：循环输出小星星**

# 控制台输出五行\*,每一行星号的数量依次递增

*\**

*\*\**

*\*\*\**

*\*\*\*\**

*\*\*\*\*\**

方法1：字符串拼接*\** 打印

# 定义一个计数器变量

row = 1

# 开始循环

**while** row <= 5:

print(**"\*"** \* row)

# 注意：计数器变化

row += 1

**4.2 print(“”,end = “”)函数的输出**

* 在默认情况下，print函数输出内容之后，会自动在内容的末尾增加换行
* 如果不希望末尾增加换行，可以在print函数输出内容的后面加上,end=””
* 其中 “” 中间可以指定print函数输出内容之后，继续希望显示的内容

print(**"\*"**, end=**""**) #不会换行 如:print(**"\*"**,end=**"---"**) *# \*---*

print(**"\*"**) # 会换行

需求：

**使用循环嵌套输出小星星**

**开发步骤：**

<1>完成5行内容的简单输出

<2>分析每行内部的\*应该如何处理？

row = 1

**while** row <= 5:

# 每一行打印的星星和行数一样

# 增加一个小循环，专门负责当前行中，每一"列"的小星星显示

# 定义一个列的计数器变量

col = 1

**"""**

**行 列**

**1 1**

**2 2**

**3 3**

**4 4**

**5 5**

**"""**

**while** col <= row:

print(**"\*"**, end=**""**)

col += 1

# print("")作用：在一行星星输出完成后，添加换行

print(**""**)

row += 1

需求:

打印九九乘法表

# 乘号前是列号，后面是行号

# 定义行的计数器

row = 1

**while** row <= 9:

# 定义列的计数器

col = 1

**while** col <= row:

# print("\*", end="")

# 列 行 乘积 用制表符\t来对齐

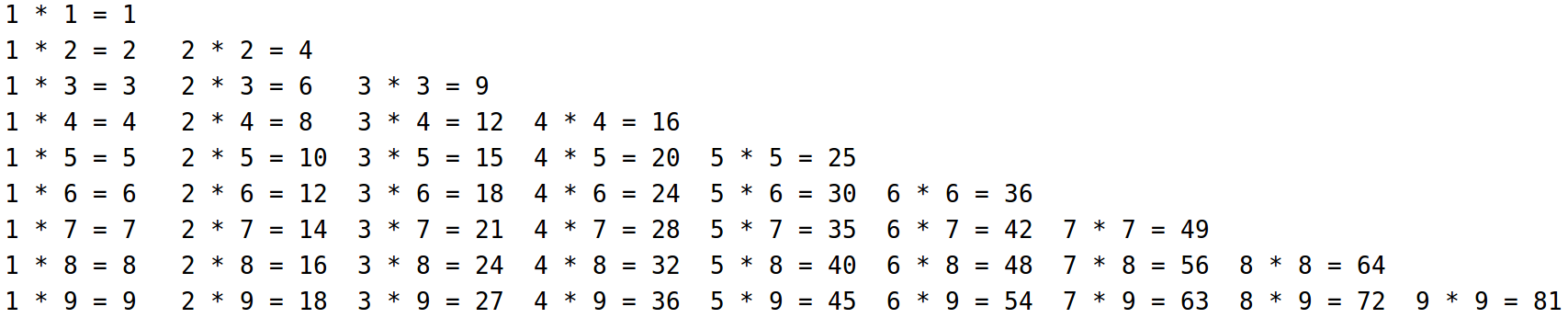
print(**"%d \* %d = %d"** % (col, row, col \* row), end=**"\t"**)

# 切记计数器加1

col += 1

print(**""**)

row += 1

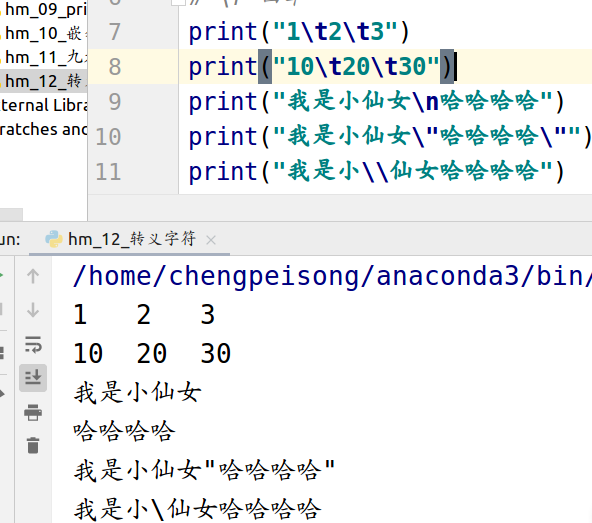


**4.3 转义字符**

**\t 制表符**  协助在输出文本时**垂直方向**保持**对齐**

**\n 换行符** 在控制台输出一个换行符

|  |  |
| --- | --- |
| 转义字符 | 描述 |
| \\ | 反斜杠符号 |
| \’ | 单引号 |
| \” | 双引号 |
| \n | 换行 |
| \t | 横向制表符 |
| \r | 回车 |

****

**234.函数的使用**

**函数的定义:**

把具有独立功能的代码块组织成一个小模块,在需要的时候调用.

**函数的使用包含两个步骤:**

1.定义函数--封装独立的功能

2.调用函数—享受封装的成果

**函数的作用:**

在开发程序时,使用函数可以提高编写的效率以及代码的**重用**

**2.1 函数的定义**

**格式:**

def 函数名():

函数封装的代码

……

**注意:**

(1)函数名称应表达函数封装代码的功能,方便后续使用

(2)函数名称的命名应符合标识符的命名规则,即:

* 可以由字母, 下划线, 数字组成
* 不能以数字开头
* 不能与关键字重名

**2.2函数的调用**

函数名()对函数进行调用

# 导入外部文件

**import** hm\_01\_九九乘法表

# 调用函数(文件名.函数名())

hm\_01\_九九乘法表.multiple\_table()

**2.3 演练1**

需求:

1.编写一个say\_hello的函数,封装三行打招呼的代码

2.在函数下方调用打招呼的代码

# 注意:定义了函数后,只表示这个函数封装了一段代码而已

# 如果不主动调用函数,函数是不会主调执行的

**def** say\_hello():

print(**"hello 1"**)

print(**"hello 2"**)

print(**"hello 3"**)

say\_hello()

**2.4改进代码**

 函数的执行线路图

2.5 函数的定义和调用顺序

**函数必须先定义,再调用(函数调用函数除外)**

****

**2.6 pycharm调试工具**

**F8 Step over** 单步调试,building把函数调用看作一行代码直接执行

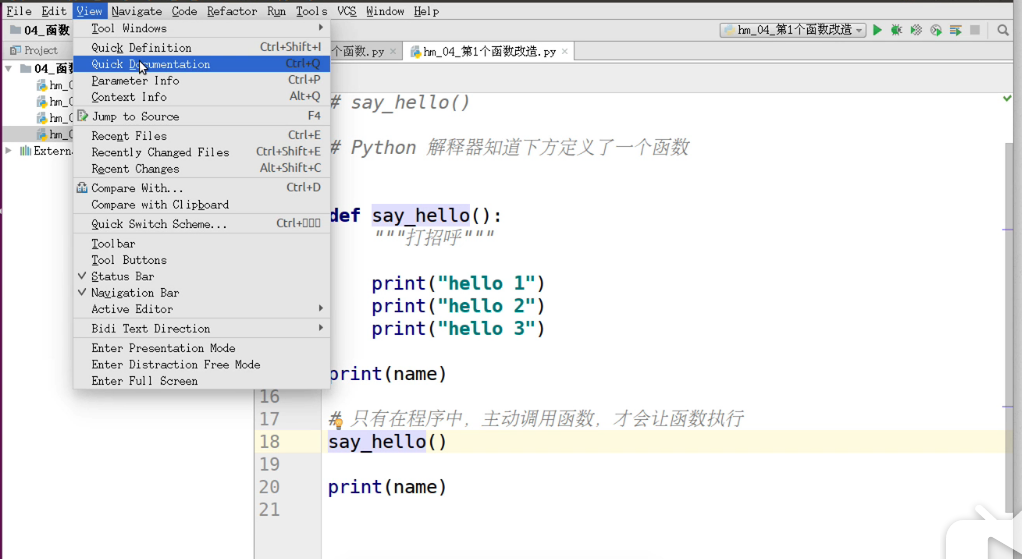
**F7 Step Into** 可以单步执行代码.**如果是函数,会进入函数内部**

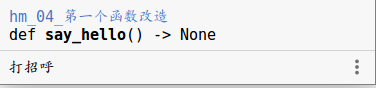
**2.7 函数的文档注释**

* **在定义函数的下方,连续使用三对引号,在三对引号处编写对函数的说明文字**
* **在函数调用位置,使用快捷键Ctrl+Q可以查看函数的说明信息**

1.函数定义**上方应该和其他代码(包含注释)保留2个空行**(PEP8的标准)

2.将光标放在函数调用内部某处,点击View--Quick Documentation(Ctrl+Q)--出现快速文档中函数的文档注释





**03.函数的参数**

需求:

1.开发一个sum\_2\_sum的函数

2.函数能够实现两个数字求和的功能

如果把需要计算的数字,在函数调用时,传递到函数内部就好了!

**3.1 函数参数的使用**

* 在函数名后面的小括号内部填写参数
* 多个参数之间用 , 分隔
* 函数调用时直接写参数值

**演练:求两个数的和:**

# 函数定义

**def** sum\_2\_num(num1, num2):

*"""对两个数求和*

**:param** *num1:*

**:param** *num2:*

**:return***: result*

*"""*

result = num1 + num2

*# 只有函数内部知道函数的返回结果*

print(**"%d + %d = %d"** % (num1, num2, result))

# 函数调用

sum\_2\_num(10, 20)

**3.2 参数的作用**

1.在函数**内部**,把参数当做**变量**使用,进行需要的数据处理

2.函数**调用**时,按照函数定义的**参数顺序**,把希望在函数内部处理的**数据**,通过参数传递

**3.3 形参和实参**

形参:**定义**函数时,小括号里的参数,告诉外部可以传递的参数数量,在函数内作为变量使用

实参:**调用**函数时,小括号里的参数,将数据按顺序**传递给函数内部使用**

**04.函数的返回值(return)**

一个函数执行结束后,告诉调用这一个结果

使用return关键字返回结果

调用函数的一方,可以使用变量来接收函数的返回结果 如:result= sum\_2\_sum(10,20)

注意:return表示返回.后续的代码都不会被执行

演练2:

**def** sum\_2\_sum(num1, num2):

"""对两个数求和"""

# 返回结果

**return** num1 + num2

# 调用函数,并使用result变量接收计算结果

# 返回值

result = sum\_2\_sum(10, 20)

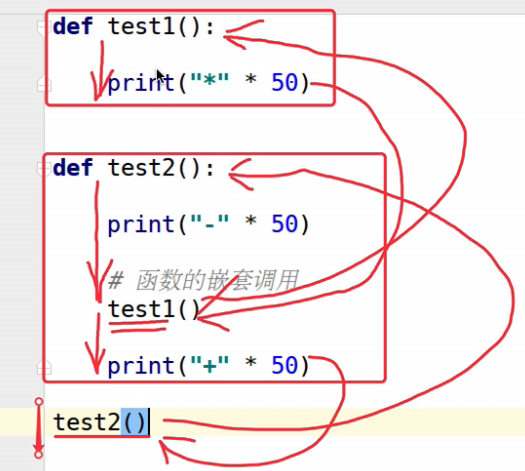
print(**"计算结果为:%d"** % result)

**249.函数的嵌套调用**

一个函数里面又调用另一个函数,这就是**函数的嵌套调用**

如果函数test2中,调用了另外一个函数test1

* 那么执行到调用test2函数时,会把函数test1中的任务都执行完
* 才会回到test2中调用函数test1的位置,继续执行后续的代码



pycharm中将run pytest改为run模式的方式:

点击菜单栏的Run-→run-→Edit congigurations

函数嵌套的演练---打印分隔线

需求1:

* 定义一个print\_line函数打印 \* 组成的一条分隔线

代码:

**def** print\_line():

print(**"\*"** \* 50)

print\_line()

需求2:

* 定义一个print\_line函数打印**任意50个字符**组成的一条分隔线

**def** print\_line(char):

print(char\* 50)

print\_line(“-”)

需求**3:**

* 定义一个**print\_line**函数打印 任意字符 任意次数的组成的一条分隔线

**def** print\_line(char,times):

print(char \* times)

print\_line(**"-"**, 10)

需求4:

* 定义一个函数能够打印5行分隔线,分隔线要求符合需求3
* **提示:**在工作中,对需求的变化,应该冷静思考,不要轻易修改之前已经完成的,能够正常执行的函数!

**def** print\_line(char,times):

print(char \* times)

**def** print\_lines():

row = 0

**while** row < 5:

print\_line(**"-"**, 50)

row += 1

print\_lines()

在封装函数函数时,要由更好的灵活性,将函数内部的数据改为参数形式,调用函数时,传递参数

**def** print\_line(char, times):

print(char \* times)

**def** print\_lines(char, times):

row = 0

**while** row < 5:

print\_line(char, times)

row += 1

print\_lines(**"-"**, 50)

**pycharm里给函数增加文档注释:**

将光标放在函数定义处,出现黄色小灯泡,点击insert documentation string stub,插入文档字符串说明,对参数和函数作用进行说明,光标放在函数调用处,按ctrl+ Q 显示快速文档说明

**06.使用模块中的函数**

**模块**是python程序架构的一个核心概念

* 模块好比是工具包,要想使用这个工具包中的工具,需要**导入import**这个模块
* 每个扩展名.py结尾的python源代码文件都是一个**模块**
* 在模块中定义的**全局变量, 函数**都是模块能够提供给外界直接使用的**工具.**

****

体验小结:

* 可以在一个python文件中定义变量或者函数
* 然后在另一个文件中使用import导入这个模块

导入之后,就可以使用**模块名.变量/模块名.函数**的方式,使用这个模块中定义的变量或者函数

**6.2 模块名也是一个标识符**

* **标识符可以由字母, 数字和下划线组成**
* **不能以数字开头**
* **不能与关键字重名**

**注意:**如果在python文件起名时,以**数字开头**是无法在pycharm中通过导入这个模块的

6.3 Pyc文件

C是compiled**编译过**的意思

操作步骤:

1.浏览程序目录会发现一个\_\_pycache\_\_的目录

2.目录下会有一个hm\_11\_分隔线模块.cpython-37.pyc 文件,cpython-37表示python解释器的版本

3. 这个pyc文件是由python解释器将**模块的源码**转换为**字节码**,python这样保存字节码是作为一种启动**速度的优化**

**字节码**

* python在解释源程序时是分成两个步骤的

1.首先处理源代码,**编译**生成一个二进制**字节码**

2.再对**字节码**进行处理,才会生成CPU能够识别的**机器码**

* 有了模块的字节码文件之后,下一次运行程序时,如果在**上次保存字节码之后**没有修改过源代码,python将会加载.pyc文件跳过编译这个步骤
* 当python重编译时,它会自动检查源文件和字节码文件的时间戳
* 如果你又修改了源代码,下次程序运行时,字节码将自动重新创建

**259.高级变量**

目标

* 列表
* 元组
* 字典
* 字符串
* 公共方法
* 变量高级

知识点回顾

数据类型: 数字型和非数字型

数字型:

* 整型(int)
* 浮点型(float)
* 布尔型(True 非0数, False 假 0)
* 复数型(complex)

非数字型

* 字符串(str)
* 列表(list)
* 字典(dict)
* 元组(tuple)

**260.列表**

01.列表的定义

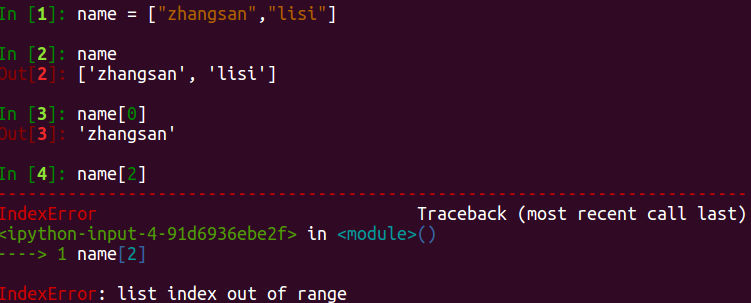
1.1 列表的定义-----列表名 = []

创建空列表

empty\_list = list()

empty\_list = []

* list(列表)是python中国使用**最频繁的数据类型**,在其他语言中通常叫做数组
* 专门用于存储一串信息
* 列表用[]定义,数据之间使用,分隔
* 列表的索引从**0**开始,索引是数据在列表中的位置编号,索引又称为**下标**
* **注意:从列表中取值时,如果超出列表的索引范围,程序会报错**

****

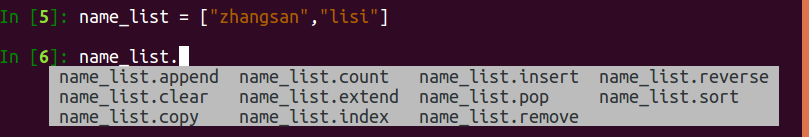
name\_list = [“zhangsan”,”lisi”,”wangwu”]

* 读取列表中的元素 name\_list[下标], i为0 ~ n
* 列表的长度 n + 1
* 统计数据在列表中出现的次数: 列表名.count(数据)

**1.2 列表的常用操作(方法)**

在ipython3中定义一个列表,如name\_list = []

**重点:在输入name\_list. 按下TAB键,ipython会提示列表可以使用的方法如下:**

(1)取值

格式: 列表名[索引]

如: name\_list = [**"zhangsan"**, **"lisi"**, **"wangwu"**]

# 1.取值-----wangwu

print(name\_list[2])

# list index out of range---列表索引超出范围

print(name\_list[3])

(2)取索引

格式: 列表名.index(元素)

如: print(name\_list.index("wangwu"))--→2

注意:如果传递的数据不在列表中,程序会报错!

print(name\_list.index(**"wangwu123"**)) ----报错

(3)修改

格式: 列表名[索引] = 对象

如: name\_list[2] = **"王五"**

print(name\_list)----->['zhangsan', 'lisi', '王五']

# 列表指定的索引超出范围,程序会报错!

name\_list[3] = **"王小二"**

(4)增加(3个)

(1) append 向列表的末尾追加数据 列表名.append(对象)

如:name\_list.append(**"小徐"**)----['zhangsan', 'lisi', 'wangwu', '小徐']

(2) insert 在列表的指定索引位置插入方法 列表名.insert(索引, 对象)

如:name\_list.insert(1, **"小兵"**)----['zhangsan', '小兵', 'lisi', 'wangwu']

(3) extend 将另一个列表中的内容追加到当前列表的末尾,将列表元素一个个的拆分开追加

列表名.extend(列表名)

如: temp\_list = [**"孙悟空"**,**"猪八戒"**,**"沙和尚"**]

name\_list.extend(temp\_list)

print(name\_list)--->['zhangsan', 'lisi', 'wangwu', '孙悟空', '猪八戒', '沙和尚']

(5)删除(3个)

(1)remove(元素) 删除列表中的指定元素

如: name\_list.remove(**"wangwu"**)--->['zhangsan', 'lisi']

(2) pop(索引) 删除列表中指定位置的元素.默认删除最后的元素

如: name\_list.remove("wangwu")-->name\_list.pop(1)---→['zhangsan', 'wangwu']

(3) clear() 清空整个列表

如: name\_list.clear()--->[]



(4)del 列表[索引] 删除指定索引的数据 del关键字,将变量从内存中删除,后续的代码不能再使用这个变量

如: name\_list = [**"张三"**, **"李四"**, **"王五"**]

# del 关键字(delete)删除列表中的元素

**del** name\_list[1]----> [**"张三"**, **"王五"**]

# del本质上是用来将一个变量从内存中删除的

name = **"小明"**

**del** name

# 注意:如果使用del关键字将变量从内存中删除

# 后续的代码就不能再使用这个变量了,

# 报错!

print(name)

(6)列表长度len()

name\_list = [**"张三"**, **"李四"**, **"王五"**, **"王小二"**, **"张三"**]

# len(列表名) 统计列表中的元素总数，列表的长度

list\_len = len(name\_list)---->5

print(**"列表中包含%d个元素"** % list\_len)--→列表中包含5个元素

(7)统计列表中某元素的个数

# count统计列表中某以一元素出现的次数

count = name\_list.count(**"张三"**)

print(**"张三在列表中出现了%d次"** % count)

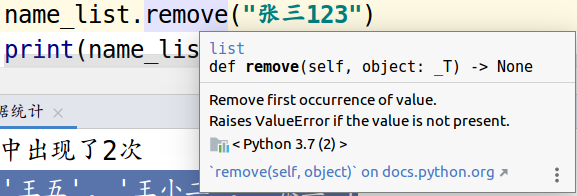
# 会删除列表中第一次出现的数据

# ['张三','李四', '王五', '王小二', '张三']

name\_list.remove(**"张三"**)--->['李四', '王五', '王小二', '张三']

print(name\_list)

按下ctrl+q查看方法的使用



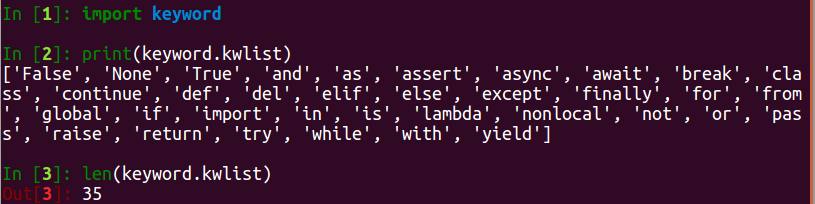
(7)排 序

列表.sort() 升序排序

列表.sort(reverse=True) 降序排序

列表.reverse() 逆序，反转

（8）补充 关键字、函数、方法的区别



* 关键字是python的内置函数，具有特殊意义的标识符，关键字后面不需要使用括号
* 函数封装了独立的功能，可以直接使用

函数名(参数)

函数需要死记硬背

* 方法和函数类似，同样是封装了独立的的功能
* 方法需要通过对象来调用，表示针对这个对象要做的操作

对象.方法名(参数)

* 在变量后面输入. ,然后针对这个变量进行要执行的操作，记忆起来比函数要简单的多

**1.3 循环遍历**

* python中为了提高列表的遍历效率，提供了迭代(iteration)遍历
* 使用for能够实现迭代遍历

**for** name **in** name\_list:

print(**"我的名字叫%s"** % name)

**1.4 列表的应用场景**

列表中可以存储不同类型的数据，



在开发中，主要用于

1.列表存储相同类型的数据

2.通过迭代遍历，在循环体内部，针对每项元素，执行相同的操作

**02. 元组**

**2.1 元组的定义**

tuple(元组)和列表类似，不同之处在于元组的元素不能修改

* 元组表示多个元素组成的序列
* 元组在python开发中，由特定的应用场景
* 用于存储一串信息，数据之间使用 , 隔开
* 元组用()定义
* 元组的索引从0开始

如：info\_tuple(“张三”,18, True,1.75)

# 查看数据类型

type(info\_tuple)--->tuple

# 查看元组中的元素

info\_tuple[1]--→18

定义空元组(一般不定义空元组，因为无法修改)

my\_tuple = ()

my\_tuple = tuple()

元组中值包含一个元素(在元素后面加个逗号)

single\_tuple = (1,)

type(single\_tuple)---→tuple

注意：

single\_tuple1 = (1)

type(single\_tuple1)---->int

**2.2元组的常用操作**

输入”元组名.”按TAB键，查看元组的方法

tuple = (**"zhangsan"**, 18, 1.75, **"zhangsan"**)

# 1.取值

print(tuple[0])

# 2.取索引

print(tuple.index(**"zhangsan"**))

# 3.统计某元素在元组中出现的次数

print(tuple.count(**"zhangsan"**))

# 4.元组的长度

print(len(tuple))

**2.3 循环遍历**

info\_tuple = (**"zhangsan"**, 18, 1.75)

# 使用迭代遍历元组

**for** my\_info **in** info\_tuple:

# 使用格式字符串拼接my\_info这个变量不方便

# 因为元组中通常保存的数据类型是不同的

**print**(my\_info)

在python中，可以使用for循环遍历的所有非数字型的变量：列表、元组、字典以及字符串

提示“在实际开发中，除非能够确认元组中的数据类型，否则针对元组的循环遍历需要不是很多

**2.4 应用场景**

可以使用for in循环遍历元组

更多应用场景：

* 函数的参数和返回值。一个函数可以接收任意多个参数，或者一次返回多个数据
* 格式化字符串，格式化字符串后面的()本质上是元组
* 让列表不可以被修改，将列表类型转换为元组类型，以保护数据安全

如：info\_tuple = ("小明", 18, 1.85)

# 格式化字符串后面的()本质上是元组

# print("%s的年龄是%d岁,身高是%.2f米" % ("小明", 18, 1.85))

print(**"%s的年龄是%d岁,身高是%.2f米"** % info\_tuple)

info\_str = **"%s的年龄是%d岁,身高是%.2f米"** % info\_tuple

**# <class 'str'>**

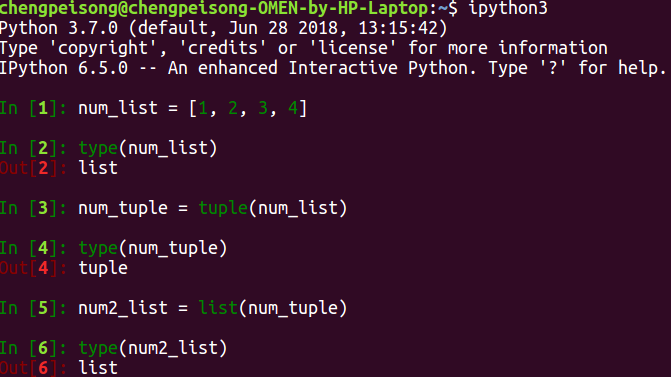
print(type(info\_str))

print(info\_str)

**2.5 元组和列表之间的转换**

list() 将元组转换为列表

tuple() 将列表转换为元组



**3.字典**

**3.1字典的定义**

字典同样用来存储多个数据

通常用来存储描述一个物体的相关信息

字典和列表的区别：

* 列表是有序的对象集合，字典是无序的对象集合
* 字典用{}来定义，列表用[]定义

字典用键值对存储数据，键值对之间用 , 隔开

* 键 key 是索引
* 值 value 是数据
* 键和值之间用 : 分隔
* 键必须是唯一的
* 值可以是任何数据类型，但 键 只能使用字符串，数字或元组
* len(字典)获取字典的键值对的数量---字典的长度
* 字典.keys()所有key列表
* 字典.values()所有value的列表
* 字典.items()所有(key,value)元组列表

**3.2 变量的定义和应用**

**注意:每个键值对各占一行**

**# 字典是无序的数据集合,使用print打印后通常输出的顺序和定义顺序不一致**

xiaoming ={**"name"**: **"小明"**,

**"age"**: 18,

**"gender"**: **True**,

**"height"**: 1.75,

**"weight"**: 56}

print(xiaoming)

**3.3 字典的常用操作**

xiaoming\_dict = {"name": "小明"}

**# 1. 取值 字典名[key]**

(1) 若key存在,直接取值

print(xiaoming\_dict["name"])

若key不存在,程序会报错!

print(xiaoming\_dict["name123"])

**2. 增加/修改 字典名[key] = value**

(1) 若key不存在,会新增键值对

xiaoming\_dict[**"age"**] = 18

(2) 若key存在,会修改已经存在的键值对

xiaoming\_dict[**"name"**] = **"小小明"**

(3) 如果key存在,不会修改数据,保持原来的值不变

如果key不存在,新建键值对

字典**.**setdefault(key,value)

如: # 若key不存在,新增键值对

xiaoming\_dict.setdefault(**"age"**,**"18"**)

# 若key存在,保持原来的值不变

xiaoming\_dict.setdefault(**"age"**,**"19"**)

**# 3.删除 字典名.pop(key)**

(1) 若key存在,则会删除对应的键值对

xiaoming\_dict.pop("name")

(2) 若key不存在,程序会报错!

xiaoming\_dict.pop("name123")

print(xiaoming\_dict)

(3) 随机删除一个键值对

字典名.popitem()

# 随机删除一对键值对

xiaoming\_dict.popitem()

(4) del 字典[key] 删除指定键值对,key不存在会报错

**del** xiaoming\_dict[**"age"**]

# 删除指定键值对,key不存在会报错

**del** xiaoming\_dict[**"age123"**]

**# 4.所有key的列表**

print(xiaoming\_dict.keys())---->dict\_keys(['name', 'age'])

**# 5.所有value列表**

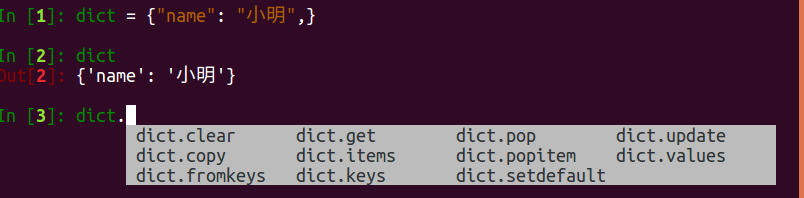
print(xiaoming\_dict.values())---->dict\_values(['小明', 18])

**# 6.所有(key,value)元组列表**

print(xiaoming\_dict.items())---->dict\_items([('name', '小明'), ('age', 18)])

**3.4 字典的其他操作**

字典名.按TAB键,可以查询字典的常用方法



xiaoming\_dict = {**"name"**: **"小明"**}

**# 1.统计键值对的数量 len(字典名)**

print(len(xiaoming\_dict))-->1

**# 2.合并两个或多个字典 字典名.update()**

temp\_dict = {**"age"**: 18}

xiaoming\_dict.update(temp\_dict)-->{'name': '小明', 'age': 18}

**# 3.清空字典 字典名.clear()**

xiaoming\_dict.clear()

print(xiaoming\_dict)--->{}

**3.5 循环遍历---for**

# 变量k是每一次循环中获得的键值对的key

**for** k **in** xiaoming\_dict:

print(**"%s - %s"** % (k, xiaoming\_dict[k]))

**3.6 应用场景**

尽管可以使用for in 循环遍历

但是在开发中,更多的应用场景是:

使用多个键值对,存储描述一个物体的相关信息---描述更复杂的数据信息

将多个字典放在一个列表中,再进行遍历,在循环体内部对每个字典进行相同的操作

card\_list =[

{"name": "小明",

"age": 18,

"gender": True},

{"name": "小丽",

"age": 15,

"gender": False}]

# 循环遍历列表

**for** card\_info **in** card\_list:

print(card\_info)

4.字符串

4.1 字符串的定义

字符串是一串字符,是编程语言中表示文本的数据类型

在python中可以用一对双引号””或一对单引号’’来定义字符串

虽然可以使用\”或者\’来做字符串的转义,但实际开发中:

* 如果字符串内部需要使用”,可以用’来定义字符串
* 如果字符串内部需要使用’,可以用”来定义字符串
* 用索引来获取字符串中指定位置的字符,索引从0开始
* 可以使用for循环来遍历字符串中的每一字符
* 大多数编程语言都是使用””来定义字符串

4.2 字符串的统计操作

hello\_str = **"hello hello"**

1. 统计字符串的长度

**print**(len(hello\_str))

2. 统计小(子)字符串出现的次数

**print**(hello\_str.count(**"llo"**))

# 若子字符串不存在,不会报错,统计次数为0

**print**(hello\_str.count(**"abc"**))

# 从开始位置2到结束位置6,子字符串出现的次数

**print**(hello\_str.count(**"llo"**, 2, 6))

3.某一子字符串出现的位置

**print**(hello\_str.index(**"llo"**))

# 从开始位置到结束位置,子字符串首次出现的位置

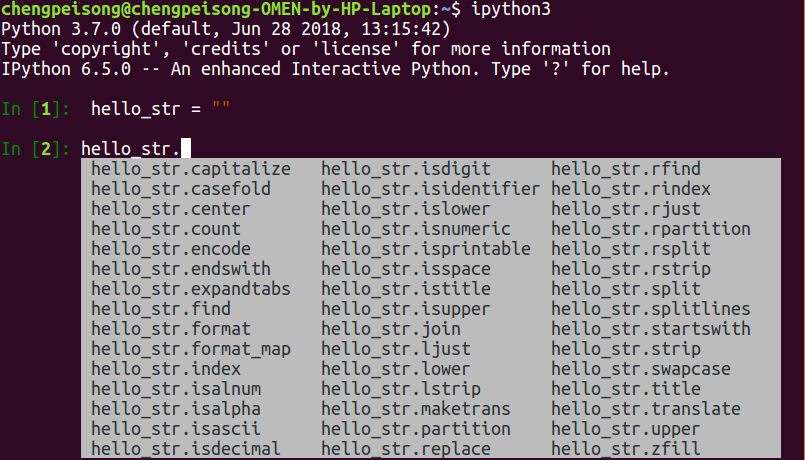
**print**(hello\_str.index(**"llo"**,1, 6))

# 若子字符串不存在,程序报错!

# print(hello\_str.index("abc"))

4.3 字符串的常用操作

在ipython中定义一个字符串,输入字符串. 按住TAB键,可以查看自付出你的所有操作

(1)判断类型(9个)

# str = " \t\n\r"

# 1. 判断是否只包含空格,则返回True

# print(str.isspace())

# 2. 判断是否字符串中的字符是否为字母或者数字,则返回True

# str = "adn35"

# print(str.isalnum())

# str = "adf"

# print(str.isalnum())

# 3.判断所有字符都是字母,则返回True

str = **"abns"**

**print**(str.isalpha())

# 4. 判断字符串中是否只包含数字(**开发中常用isdecimal()**)

# num\_str = "1.8"

# <1>isdecimal(),isdigit(),isnumeric()都可以判断整型数字,都不能判断小数

# print(num\_str.isdecimal())

# print(num\_str.isdigit())

# print(num\_str.isnumeric())

# <2>isdecimal()不可以判断unicode字符串

# num\_str = "\u00b2"

# print(num\_str) #---->平方符号

# print(num\_str.isdecimal()) #---->False

# print(num\_str.isdigit()) #---->True

# print(num\_str.isnumeric()) #---->True

# <3>isnumeric()可以判断汉字中的数字

num\_str = **"一千零一"**

**print**(num\_str)

**print**(num\_str.isdecimal()) #--->False

**print**(num\_str.isdigit()) #--->False

**print**(num\_str.isnumeric()) #--->True

(2)查找和替换(7个)

****

**hello\_str = "hello world"**

# 1.判断是否以指定的字符串开头

# 返回True

# print(hello\_str.startswith("h"))

# 2.判断是否以指定的字符串结尾

# print(hello\_str.endswith("d"))

# 3.查找指定的字符串

# find()和index()都可以查找指定字符串在大字符串中的索引

# find()如果指定的字符串不存在,会返回-1

# print(hello\_str.find("abc"))

# 从右开始寻找,返回从左数的下标索引-->6

# print(hello\_str.rfind("w"))

# index()如果指定的字符串不存在,会报错!

# print(hello\_str.index("abc"))

# 4.替换字符串

# replace方法执行完成后,会返回一个新的字符串

# 注意:不会修改原来字符串的内容

**# print**(hello\_str.replace(**"world"**, **"python"**))

# hello world

**# print**(hello\_str)

(3)大小写转换(5个)



(4)文本对齐(3个)

# 要求:顺序并且居中对齐输出以下内容

poem = [**"登鹳雀楼"**,

**"王之涣"**,

**"白日依山尽"**,

**"黄河入海流"**,

**"欲穷千里目"**,

**"更上一层楼"**]

**for** poem\_str **in** poem:

# "　"处要用中文全角字体输入，10表示width

# "　"表示fillchar填充字符

**print**(**"|%s|"** % poem\_str.center(10, **"　"**))

# print("|%s|" % poem\_str.ljust(10, "　"))

# print("|%s|" % poem\_str.rjust(10, "　"))

(5)去除空白字符(3个)



poem = [**"\n\t登鹳雀楼"**,

**"王之涣"**,

**"白日依山尽\t"**,

**"黄河入海流"**,

**"欲穷千里目\t"**,

**"更上一层楼"**]

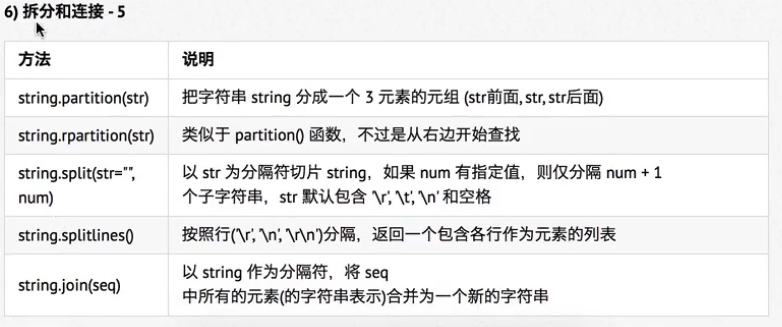
**for** poem\_str **in** poem:

**# 先使用strip()去除字符串中的空白字符**

# 然后使用center函数居中显示文本

**print**(**"|%s|"** % poem\_str.strip().center(10, **"　"**))

(6)拆分和拼接(5个)

**poem\_str = "登鹳雀楼\t王之涣\t白日依山尽\t黄河入海流\t欲穷千里目\t更上一层楼"**

# 1.去掉空白字符(\t\n\r 空格),返回列表类型

**print**(poem\_str.split())

# 2.合并字符串

# 使用" "作为分隔符，拼接字符串,返回字符串类型

result = **" "**.join(poem\_str)

**print**(result)

# print(poem\_str)

4.4 字符串的切片

切片适用于字符串, 列表, 元组

切片使用索引值来限定范围,从一个大的字符串中切除晓得字符串

列表和元组都是有序的集合,都能够通过索引值获得对应的数据

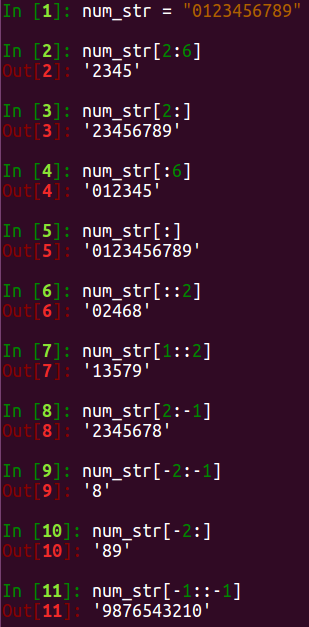
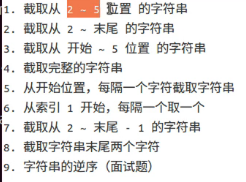
字典是无序的集合,使用键值对保存数据

正序(0,1,2) 倒序(-1,-2 )

* 切片的格式: 字符串[开始索引:结束索引:步长]
* 不包括结束索引位置的元素,
* 要想遍历所有字符串,将结束索引置为空

4.5 切片演练

需求:



**字符串的逆序(面试题)**

**num\_str[-1::-1]**

5. 公共方法

5.1 Python内置函数

a = [1,2,3]

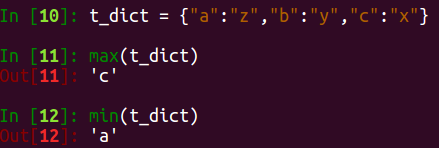
len(item) 计算容器中元素的个数 如: len(a)--->3

del(item) 删除变量 有两种方式 如: del(a) del a[1]

max(item) 返回容器中的最大值 如: max(a)---→3

min(item) 返回容器中元素的最小值 如: min(a)---->1

对字典进行比较时,只比较key,不比较value



cmp(item1,item2) 比较两个值 python3.x中取消了cmp函数

注意: 字符串比较 0 < “A” < “a”

要想比较列表,字符串,元组,需要使用比较运算符,字典不能进行比较

