# python

## 注释方法

"""

第一个Python程序 - hello, world!

向伟大的Dennis M. Ritchie先生致敬

"""

# print("你好,世界！")

## 语言元素

* int()：将一个数值或字符串转换成整数，可以指定进制。
* float()：将一个字符串转换成浮点数。
* str()：将指定的对象转换成字符串形式，可以指定编码。
* chr()：将整数转换成该编码对应的字符串（一个字符）。
* ord()：将字符串（一个字符）转换成对应的编码（整数）。



## for-in循环

sum = 0

for x in range(101):

sum += x

print(sum)

range(101)可以产生一个0到100的整数序列。

range(1, 100)可以产生一个1到99的整数序列。

range(1, 100, 2)可以产生一个1到99的奇数序列，其中的2是步长，即数值序列的增量。

len(str)

li =list (map(int,input("plz input num:")))

## while循环

import random

answer = random.randint(1, 100)

counter = 0

while True:

counter += 1

number = int(input('请输入: '))

if number < answer:

print('大一点')

elif number > answer:

print('小一点')

else:

print('恭喜你猜对了!')

break

print('你总共猜了%d次' % counter)

if counter > 7:

print('你的智商余额明显不足')

## 字符串和常用数据结构

str1 = 'hello, world!'

print(len(str1)) # 13

print(str1.capitalize()) # Hello, world!

print(str1.upper()) # HELLO, WORLD!

print(str1.find('or')) # 8

print(str1.find('shit')) # -1

# 检查字符串是否以指定的字符串开头

print(str1.startswith('He')) # False

print(str1.startswith('hel')) # True

# 检查字符串是否以指定的字符串结尾

print(str1.endswith('!')) # True

# 将字符串以指定的宽度居中并在两侧填充指定的字符

print(str1.center(50, '\*'))

# 将字符串以指定的宽度靠右放置左侧填充指定的字符

print(str1.rjust(50, ' '))

str2 = 'abc123456'

# 从字符串中取出指定位置的字符(下标运算)

print(str2[2]) # c

# 字符串切片(从指定的开始索引到指定的结束索引)

print(str2[2:5]) # c12

print(str2[2:]) # c123456

print(str2[2::2]) # c246

print(str2[::2]) # ac246

print(str2[::-1]) # 654321cba

print(str2[-3:-1]) # 45

# 检查字符串是否由数字构成

print(str2.isdigit()) # False

# 检查字符串是否以字母构成

print(str2.isalpha()) # False

# 检查字符串是否以数字和字母构成

print(str2.isalnum()) # True

str3 = ' jackfrued@126.com '

print(str3)

# 获得字符串修剪左右两侧空格的拷贝

print(str3.strip())

| **函数** | **说明** |
| --- | --- |
| compile(pattern, flags=0) | 编译正则表达式返回正则表达式对象 |
| match(pattern, string, flags=0) | 用正则表达式匹配字符串 成功返回匹配对象 否则返回None |
| search(pattern, string, flags=0) | 搜索字符串中第一次出现正则表达式的模式 成功返回匹配对象 否则返回None |
| split(pattern, string, maxsplit=0, flags=0) | 用正则表达式指定的模式分隔符拆分字符串 返回列表 |
| sub(pattern, repl, string, count=0, flags=0) | 用指定的字符串替换原字符串中与正则表达式匹配的模式 可以用count指定替换的次数 |
| fullmatch(pattern, string, flags=0) | match函数的完全匹配（从字符串开头到结尾）版本 |
| findall(pattern, string, flags=0) | 查找字符串所有与正则表达式匹配的模式 返回字符串的列表 |
| finditer(pattern, string, flags=0) | 查找字符串所有与正则表达式匹配的模式 返回一个迭代器 |
| purge() | 清除隐式编译的正则表达式的缓存 |
| re.I / re.IGNORECASE | 忽略大小写匹配标记 |
| re.M / re.MULTILINE | 多行匹配标记 |

# 创建正则表达式对象 使用了前瞻和回顾来保证手机号前后不应该出现数字

pattern = re.compile(r'(?<=\D)1[34578]\d{9}(?=\D)')

sentence = '''

重要的事情说8130123456789遍，我的手机号是13512346789这个靓号，

不是15600998765，也是110或119，王大锤的手机号才是15600998765。

'''

# 查找所有匹配并保存到一个列表中

mylist = re.findall(pattern, sentence)

print(mylist)

# 通过迭代器取出匹配对象并获得匹配的内容

for temp in pattern.finditer(sentence):

print(temp.group())

# 通过search函数指定搜索位置找出所有匹配

m = pattern.search(sentence)

while m:

print(m.group())

m = pattern.search(sentence, m.end())

## 列表和元组

列表就是数组，元组是不可更改只能替换的数组。

def main():

# 定义元组

t = ('骆昊', 38, True, '四川成都')

# 创建空列表

t=[]

print(t)

# 替换元组

t = ('王大锤', 20, True, '云南昆明')

print(t)

# 将元组转换成列表

person = list(t)

print(person)

# 列表是可以修改它的元素的

# 将列表转换成元组

fruits\_list = ['apple', 'banana', 'orange']

fruits\_tuple = tuple(fruits\_list)

print(fruits\_tuple)

## 集合

set1 = {1, 2, 3, 3, 3, 2}

print(set1)

## 字典

def main():

scores = {'骆昊': 95, '白元芳': 78, '狄仁杰': 82}

# 通过键可以获取字典中对应的值

print(scores['骆昊'])

print(scores['狄仁杰'])

# 对字典进行遍历(遍历的其实是键再通过键取对应的值)

for elem in scores:

print('%s\t--->\t%d' % (elem, scores[elem]))

# 更新字典中的元素

scores['白元芳'] = 65

scores['诸葛王朗'] = 71

scores.update(冷面=67, 方启鹤=85)

print(scores)

if '武则天' in scores:

print(scores['武则天'])

print(scores.get('武则天'))

# get方法也是通过键获取对应的值但是可以设置默认值

print(scores.get('武则天', 60))

# 删除字典中的元素

print(scores.popitem())

print(scores.popitem())

print(scores.pop('骆昊', 100))

# 清空字典

scores.clear()

print(scores)

## input

a,b=map(int,input().split())

print('yi',end=" ")

.format(5)

list=[]

list= (input().split())

# 类

## 定义类

class Student(object):

# \_\_init\_\_是一个特殊方法用于在创建对象时进行初始化操作

# 通过这个方法我们可以为学生对象绑定name和age两个属性

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def study(self, course\_name):

print('%s正在学习%s.' % (self.name, course\_name))

# PEP 8要求标识符的名字用全小写多个单词用下划线连接

# 但是部分程序员和公司更倾向于使用驼峰命名法(驼峰标识)

def watch\_movie(self):

if self.age < 18:

print('%s只能观看《熊出没》.' % self.name)

else:

print('%s正在观看岛国爱情大电影.' % self.name)

## 创建和使用对象

def main():

# 创建学生对象并指定姓名和年龄

stu1 = Student('骆昊', 38)

# 给对象发study消息

stu1.study('Python程序设计')

# 给对象发watch\_av消息

stu1.watch\_movie()

stu2 = Student('王大锤', 15)

stu2.study('思想品德')

stu2.watch\_movie()

## 访问可见性问题

class Test:

def \_\_init\_\_(self, foo):

#两个下划线开头是private

self.\_\_foo = foo

def \_\_bar(self):

print(self.\_\_foo)

print('\_\_bar')

def main():

test = Test('hello')

test.\_Test\_\_bar()

print(test.\_Test\_\_foo)

## @property装饰器

class Person(object):

def \_\_init\_\_(self, name, age):

self.\_name = name

self.\_age = age

# 访问器 - getter方法

@property

def name(self):

return self.\_name

# 访问器 - getter方法

@property

def age(self):

return self.\_age

# 修改器 - setter方法

@age.setter

def age(self, age):

self.\_age = age

def play(self):

if self.\_age <= 16:

print('%s正在玩飞行棋.' % self.\_name)

else:

print('%s正在玩斗地主.' % self.\_name)

def main():

person = Person('王大锤', 12)

person.play()

person.age = 22

person.play()

# person.name = '白元芳' # AttributeError: can't set attribute

### \_\_slots\_\_魔法

class Person(object):

# 限定Person对象只能绑定\_name, \_age和\_gender属性

\_\_slots\_\_ = ('\_name', '\_age', '\_gender')

def \_\_init\_\_(self, name, age):

self.\_name = name

self.\_age = age

@property

def name(self):

return self.\_name

@property

def age(self):

return self.\_age

@age.setter

def age(self, age):

self.\_age = age

def play(self):

if self.\_age <= 16:

print('%s正在玩飞行棋.' % self.\_name)

else:

print('%s正在玩斗地主.' % self.\_name)

def main():

person = Person('王大锤', 22)

person.play()

person.\_gender = '男'

# AttributeError: 'Person' object has no attribute '\_is\_gay'

# person.\_is\_gay = True

### 静态方法和类方法

from math import sqrt

class Triangle(object):

def \_\_init\_\_(self, a, b, c):

self.\_a = a

self.\_b = b

self.\_c = c

@staticmethod

def is\_valid(a, b, c):

return a + b > c and b + c > a and a + c > b

def perimeter(self):

return self.\_a + self.\_b + self.\_c

def area(self):

half = self.perimeter() / 2

return sqrt(half \* (half - self.\_a) \*

(half - self.\_b) \* (half - self.\_c))

def main():

a, b, c = 3, 4, 5

# 静态方法和类方法都是通过给类发消息来调用的

if Triangle.is\_valid(a, b, c):

t = Triangle(a, b, c)

print(t.perimeter())

# 也可以通过给类发消息来调用对象方法但是要传入接收消息的对象作为参数

# print(Triangle.perimeter(t))

print(t.area())

# print(Triangle.area(t))

else:

print('无法构成三角形.')

if \_\_name\_\_ == '\_\_main\_\_':

main()

# GUI

## 基于tkinter模块的GUI

import tkinter

import tkinter.messagebox

def main():

flag = True

# 修改标签上的文字

def change\_label\_text():

nonlocal flag

flag = not flag

color, msg = ('red', 'Hello, world!')\

if flag else ('blue', 'Goodbye, world!')

label.config(text=msg, fg=color)

# 确认退出

def confirm\_to\_quit():

if tkinter.messagebox.askokcancel('温馨提示', '确定要退出吗?'):

top.quit()

# 创建顶层窗口

top = tkinter.Tk()

# 设置窗口大小

top.geometry('240x160')

# 设置窗口标题

top.title('小游戏')

# 创建标签对象并添加到顶层窗口

label = tkinter.Label(top, text='Hello, world!', font='Arial -32', fg='red')

label.pack(expand=1)

# 创建一个装按钮的容器

panel = tkinter.Frame(top)

# 创建按钮对象 指定添加到哪个容器中 通过command参数绑定事件回调函数

button1 = tkinter.Button(panel, text='修改', command=change\_label\_text)

button1.pack(side='left')

button2 = tkinter.Button(panel, text='退出', command=confirm\_to\_quit)

button2.pack(side='right')

panel.pack(side='bottom')

# 开启主事件循环

tkinter.mainloop()

if \_\_name\_\_ == '\_\_main\_\_':

main()

## pygame

import pygame

def main():

# 初始化导入的pygame中的模块

pygame.init()

# 初始化用于显示的窗口并设置窗口尺寸

screen = pygame.display.set\_mode((800, 600))

# 设置当前窗口的标题

pygame.display.set\_caption('大球吃小球')

running = True

# 开启一个事件循环处理发生的事件

while running:

# 从消息队列中获取事件并对事件进行处理

for event in pygame.event.get():

if event.type == pygame.QUIT:

running = False

if \_\_name\_\_ == '\_\_main\_\_':

main()

# 文件

| **操作模式** | **具体含义** |
| --- | --- |
| 'r' | 读取 （默认） |
| 'w' | 写入（会先截断之前的内容） |
| 'x' | 写入，如果文件已经存在会产生异常 |
| 'a' | 追加，将内容写入到已有文件的末尾 |
| 'b' | 二进制模式 |
| 't' | 文本模式（默认） |
| '+' | 更新（既可以读又可以写） |

f = None

# 那些在运行时可能出现的代码放在try代码块中

# 跟上except来捕获可能出现的异常状况

try:

f = open('致橡树.txt', 'r', encoding='utf-8')

print(f.read())

except FileNotFoundError:

print('无法打开指定的文件!')

except LookupError:

print('指定了未知的编码!')

except UnicodeDecodeError:

print('读取文件时解码错误!')

finally:

if f:

f.close()

# 一次性读取整个文件内容

with open('致橡树.txt', 'r', encoding='utf-8') as f:

print(f.read())

# 通过for-in循环逐行读取

with open('致橡树.txt', mode='r') as f:

for line in f:

print(line, end='')

time.sleep(0.5)

print()

# 读取文件按行读取到列表中

with open('致橡树.txt') as f:

lines = f.readlines()

print(lines)

# 算法

## 简单选择排序

def select\_sort(origin\_items, comp=lambda x, y: x < y):

"""简单选择排序"""

items = origin\_items[:]

for i in range(len(items) - 1):

min\_index = i

for j in range(i + 1, len(items)):

if comp(items[j], items[min\_index]):

min\_index = j

items[i], items[min\_index] = items[min\_index], items[i]

return items

## 冒泡排序

def bubble\_sort(origin\_items, comp=lambda x, y: x > y):

"""高质量冒泡排序(搅拌排序)"""

items = origin\_items[:]

for i in range(len(items) - 1):

swapped = False

for j in range(i, len(items) - 1 - i):

if comp(items[j], items[j + 1]):

items[j], items[j + 1] = items[j + 1], items[j]

swapped = True

if swapped:

swapped = False

for j in range(len(items) - 2 - i, i, -1):

if comp(items[j - 1], items[j]):

items[j], items[j - 1] = items[j - 1], items[j]

swapped = True

if not swapped:

break

return items

## 归并排序(分治法)

def merge\_sort(items, comp=lambda x, y: x <= y):

"""归并排序(分治法)"""

if len(items) < 2:

return items[:]

mid = len(items) // 2

left = merge\_sort(items[:mid], comp)

right = merge\_sort(items[mid:], comp)

return merge(left, right, comp)

def merge(items1, items2, comp):

"""合并(将两个有序的列表合并成一个有序的列表)"""

items = []

index, index2 = 0, 0

while index1 < len(items1) and index2 < len(items2):

if comp(items1[index1], items2[index2]):

items.append(items1[index1])

index1 += 1

else:

items.append(items2[index2])

index2 += 1

items += items1[index1:]

items += items2[index2:]

return items