**Python猜数小游戏**

【游戏规则】生成一个指定范围的随机数（如：1-100），然后玩家输入数值猜答案，屏幕会根据玩家输入的数字给出大小提示，一直到玩家猜出准确答案则游戏胜利并结束。

import random

answer=random.**randint**(1,100)

n=**int**(input("Please input num: "))

while n!=answer :

**if** n>answer:

n = **int**(input("Num is Big! Please Continue input: "))

elif n<answer:

n = **int**(input("Num is small! Please Continue input: "))

**print**("You Win the game!")

**Python 函数**

**函数概念**

函数是组织好的，可重复使用的，用来实现单一，或相关联功能的代码段。 函数能提高应用的模块性，和代码的重复利用率。如print() range()函数，但你也可以自己创建函数，这被叫做用户自定义函数。

**函数定义**

案例：定义一个函数Max\_num()，用来比较两个数字的大小，然后将数值大的数字返回。

* 函数代码块以 def 关键词开头，后接函数标识符名称和圆括号()。
* 圆括号之间可以用于定义参数。
* 函数内容以冒号起始，并且缩进。
* return [表达式] 选择性地返回一个值给调用方。不带表达式的return相当于返回 None。

def **Max\_num** (a,b):

**if** a>b:

**return** a

**elif** a<b:

**return** b

**else**:

**return** a

result=**Max\_num**(15,10)

print(result)

**Python 面向对象**

### 场景案例：

在自学网班级要求两名新同学Jack和Harry分别介绍自己的名字和来自哪座城市，然后分别说一句班级口号：“Hello,51zxw” 最终控制台打印效果如下：

My name **is** Jack **and** come **from** Beijing

Hello 51zxw!

My name **is** Harry **and** come **from** Shanghai

Hello 51zxw!

仅结合前面所学基础知识来实现：

name='Jack'

city='Beijing'

**print**("My name is %s and come from %s" %(name,city))

**print**("Hello 51zxw!")

name='Harry'

city='Shanghai'

**print**("My name is %s and come from %s" %(name,city))

**print**("Hello 51zxw!")

**？思考几个问题？**

1. 如果老师要求全班50个同学依次以上面形式自我介绍，该怎么办？
2. 每个同学介绍自己姓名和来自城市之后，再顺便介绍自己的年龄？
3. 每个同学自我介绍的代码块有何相同特征？

### 类与对象

现实世界中，随处可见的一种事物就是对象，对象是事物存在的实体，如人类、汽车、动物、水果这些都是一个抽象的类别，我们所见到的实物都是这些类的具体存在，因此**类是对象的抽象集合，对象是类的具体表现**。现实世界是万物皆对象！

人

* 属性：地域、肤色、国家.....
* 功能：走路、思考、饮食、繁衍、....
* 具体对象：中国人，非洲人

学生

* 属性：姓名，学号、城市、年龄....
* 功能：听，说、读、写...
* 具体对象：Jack同学，Harry同学

#### 面向对象关键要素

##### 类(Class):

用来描述具有相同的属性和方法的对象的集合。它定义了该集合中每个对象所共有的属性和方法。

##### 数据成员：

类的不同属性数据。

##### 对象：

对象是类的实例

##### 方法：

类中定义的函数，实现相关的功能。

#### 面向对象编程

简称OOP（Object Oriented Programming），是一种程序设计思想。OOP把对象作为程序的基本单元，一个对象包含了数据和操作数据的函数（方法）。

面向对象的设计思想是从自然界中来的，因为在自然界中，类（Class）和实例（Instance）的概念是很自然的。Class是一种抽象概念，比如我们定义的 Student类，是指学生这个概念，而实例（Instance）则是一个个体的Student对象。

Python是一门面向对象的语言，在Python中创建一个类和对象是很容易的。

#### 三大特性

* 封装
* 继承
* 多态

### Python 类与对象

#### 定义类

**class** **Student**(**object**):

类体

Class是类的定义的关键词，class后面紧接着是类名，即Student，类名通常是大写开头的单词，紧接着是(object)，表示该类是从哪个类继承下来的。通常如果没有明确的继承类，就使用object类，括号内一般为空默认就是继承Obejct类。这是所有类最终都会继承的类，也就是基类。

#### 属性初始化

由于类可以起到模板的作用，因此，可以在创建实例对象的时候，把一些我们认为必须绑定的属性强制填写进去。通过定义一个特殊的\_\_init\_\_方法，如：在创建Student实例的时候，就把name，city等属性绑上去：

**class** **Student**():

**def** **\_\_init\_\_**(self,name,city):

self.name=name

self.city=city

print("My name is %s and come from %s" % (name, city))

\_\_init\_\_方法的第一个参数永远是self，表示创建的实例本身，因此，在\_\_init\_\_方法内部，就可以把各种属性绑定到self，因为self就指向创建的实例本身。有了\_\_init\_\_方法，在创建实例的时候，就不能传入空的参数了，必须传入与\_\_init\_\_方法匹配的参数，但self不需要传，Python解释器自己会把实例变量传进去。

#### 定义方法

类的方法除了第一个参数是self外，其他和普通函数一样。要调用一个方法，只需要在实例变量上直接调用，

**class** **Student**():

**def** **\_\_init\_\_**(self,name,city):

self.name=name

self.city=city

print("My name is %s and come from %s" % (name, city))

**def** **talk**(self):

print("Hello,51zxw")

#### 生成实例对象

stu1=**Student**('Jack','Beijing')

stu1.**talk**()

stu2=**Student**('Harry','Shanghai')

stu2.**talk**()

## 模 块

*为何要使用模块？*

随着项目功能和需求增多，代码量也会增大，把全部代码放在一个文件会显得冗余，因此需要使用模块进行分区管理。

Python模块是什么？

Python 模块(Module)，是一个 Python 文件，以 .py 结尾，包含了 Python 对象定义和Python语句。

*使用模块有什么好处？*

最大的好处是大大提高了代码的可维护性。其次，编写代码不必从零开始。当一个模块编写完毕，就可以被其他地方引用。如：随机数模块，时间模块。

#### import语句

导入时间模块显示当前系统时间

*#模块 显示当前时间*

import **time**

**print**(**time**.ctime()) *#调用获取当前时间的方法*

导入随机数模块显示随机整数

import random

num=random.**randint**()

**print**(num)

### from ...import ....

Python 的 from 语句让你从模块中导入一个指定的部分到当前命名空间中。

from time import sleep

from Student import Student

stu1=**Student**('jack','Beijing')

stu1.**talk**()

stu2=**Student**('Harry','Shanghai')

stu2.**talk**()

**跨目录调用模块**

案例：调用School目录下的Student模块

from School.Student import Student

stu1=**Student**('jack','Beijing')

stu1.**talk**()

stu2=**Student**('Harry','Shanghai')

stu2.**talk**()

**import 搜索路径**

当你导入一个模块，Python 解析器对模块位置的搜索顺序是：

* 1、当前目录
* 2、如果不在当前目录，Python 则搜索 PYTHONPATH 下的每个目录。
* 3、如果都找不到，Python会察看安装默认路径。

Python异常

## 什么是异常？

异常即是一个事件，该事件会在程序执行过程中发生，影响了程序的正常执行。 一般情况下，在Python无法正常处理程序时就会发生一个异常。 异常是Python对象，表示一个错误。 当Python脚本发生异常时我们需要捕获处理它，否则程序会终止执行。

## 常见异常类型

| **异常名称** | **描述** |
| --- | --- |
| FileNotFoundError | 找不到指定文件的异常 |
| NameError | 未声明/初始化对象 (没有属性) |
| BaseException | 所有异常的基类 |

## 异常处理语句

* try...except...
* try...except...finally
* raise

## 1.try....except

#### FileNotFoundError

try:

fileName=**input**("Please input fileName:")

**open**("%s.txt" %fileName)

except FileNotFoundError:

**print**("%s file not found " %fileName)

#### NameError

try:

**print**(stu)

except NameError:

**print**("stu not define !")

#### BaseException

try:

**print**(stu)

except BaseException:

**print**("stu not define !")

#### try…except…as…

try:

#stu='Jack'

**print**(stu)

except BaseException as msg:

**print**(msg)

### try... except... else 使用

try:

stu='Jack'

**print**(stu)

except BaseException as msg:

**print**(msg)

**else**:

**print**("stu is defined！")

### try..except...finally 输出

try:

#stu='Jack'

**print**(stu)

except BaseException as msg:

**print**(msg)

finally:

**print**("The end !")

### raise 抛出异常

### C:\Users\Shuqing\AppData\Local\Microsoft\Windows\INetCache\Content.Word\image.jfif

### 前面try语句是执行过程中捕获代码块的异常，而raise是通过事先定义一个条件，一旦符合异常条件就抛出异常。

**def** **division**(x,y):

**if** y==0:

**raise** ZeroDivisionError("Zero is not allow!")

**return** x/y

**try**:

division(8,0)

**except** BaseException **as** msg:

print(msg)



注意：raise只能用于Python标准异常类！

|  |  |
| --- | --- |
| **异常名称** | **描述** |
|  |  |
| BaseException | 所有异常的基类 |
| SystemExit | 解释器请求退出 |
| KeyboardInterrupt | 用户中断执行(通常是输入^C) |
| Exception | 常规错误的基类 |
| StopIteration | 迭代器没有更多的值 |
| GeneratorExit | 生成器(generator)发生异常来通知退出 |
| StandardError | 所有的内建标准异常的基类 |
| ArithmeticError | 所有数值计算错误的基类 |
| FloatingPointError | 浮点计算错误 |
| OverflowError | 数值运算超出最大限制 |
| **ZeroDivisionError** | 除(或取模)零 (所有数据类型) |
| AssertionError | 断言语句失败 |
| AttributeError | 对象没有这个属性 |
| EOFError | 没有内建输入,到达EOF 标记 |
| EnvironmentError | 操作系统错误的基类 |
| IOError | 输入/输出操作失败 |
| OSError | 操作系统错误 |
| WindowsError | 系统调用失败 |
| ImportError | 导入模块/对象失败 |
| LookupError | 无效数据查询的基类 |
| IndexError | 序列中没有此索引(index) |
| KeyError | 映射中没有这个键 |
| MemoryError | 内存溢出错误(对于Python 解释器不是致命的) |
| NameError | 未声明/初始化对象 (没有属性) |
| UnboundLocalError | 访问未初始化的本地变量 |
| ReferenceError | 弱引用(Weak reference)试图访问已经垃圾回收了的对象 |
| RuntimeError | 一般的运行时错误 |
| NotImplementedError | 尚未实现的方法 |
| SyntaxError | Python 语法错误 |
| IndentationError | 缩进错误 |
| TabError | Tab 和空格混用 |
| SystemError | 一般的解释器系统错误 |
| TypeError | 对类型无效的操作 |
| ValueError | 传入无效的参数 |
| UnicodeError | Unicode 相关的错误 |
| UnicodeDecodeError | Unicode 解码时的错误 |
| UnicodeEncodeError | Unicode 编码时错误 |
| UnicodeTranslateError | Unicode 转换时错误 |
| Warning | 警告的基类 |
| DeprecationWarning | 关于被弃用的特征的警告 |
| FutureWarning | 关于构造将来语义会有改变的警告 |
| OverflowWarning | 旧的关于自动提升为长整型(long)的警告 |
| PendingDeprecationWarning | 关于特性将会被废弃的警告 |
| RuntimeWarning | 可疑的运行时行为(runtime behavior)的警告 |
| SyntaxWarning | 可疑的语法的警告 |
| UserWarning | 用户代码生成的警告 |

文件处理

### 打开文件

使用Python内置的方法 open（）可以打开文件

file object = open(file\_name [, access\_mode][, buffering])

* file\_name：file\_name变量是一个包含了你要访问的文件名称的字符串值。
* access\_mode：access\_mode决定了打开文件的模式：只读，写入，追加等。所有可取值见如下的完全列表。这个参数是非强制的，默认文件访问模式为只读(r)。
* buffering:如果buffering的值被设为0，就不会有寄存。如果buffering的值取1，访问文件时会寄存行。如果将buffering的值设为大于1的整数，表明了这就是的寄存区的缓冲大小。如果取负值，寄存区的缓冲大小则为系统默认。

f=**open**('stu\_info.txt','r')

f=**open**('E:\\test\\stu\_info.txt','r')

#### 常用文件打开模式

| **模式** | **描述** |
| --- | --- |
| r | 以只读方式打开文件。 |
| rb | 以二进制格式打开一个文件用于只读。 |
| w | 打开一个文件只用于写入。 |
| a | 打开一个文件用于追加。新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。 |

### 文件读取

line=f.**read**()

line1=f.**readline**()

line2=f.**readlines**()

* read() 每次读取整个文件，它通常用于将文件内容放到一个字符串变量中。
* readline() 每次只读取一行
* readlines()一次性读取文件所有行 自动将文件内容分析成一个行的列表，该列表可以由 Python 的 for ... in ... 结构进行处理。

### 关闭文件

f.close（）

#### 读取txt文件

案例：读取stu\_info.txt文件内容，并将所有文件中学生名称显示出来

f=**open**('stu\_info.txt','r')

lines=f.**readlines**()

**print**(lines)

**for** line **in** lines:

**print**(line.split(',')[0])

#### split()方法语法：

**str**.split(**str**="", num=string.**count**(**str**)).

参数

* str -- 分隔符，默认为所有的空字符，包括空格、换行(\n)、制表符(\t)等。
* num -- 分割次数

### 读写csv文件

csv即为逗号分隔值（Comma-Separated Values，CSV），有时也称为字符分隔值，因为分隔字符也可以不是逗号），其文件以纯文本形式存储表格数据（数字和文本）。

#### csv文件读取

案例：读取Stu\_info.csv文件里所有学生信息。

import csv

csv\_file=csv.**reader**(open('Stu\_info.csv','r'))

**for** stu **in** csv\_file:

**print**(stu)

#### csv文件写入

对Stu\_info.csv文件追加写入两个学生信息

stu=['Marry',28,'Changsha']

stu1=['Rom',23,'Chengdu']

out=**open**('Stu\_info.csv','a',newline='')

csv\_write=csv.**writer**(out,dialect='excel')

csv\_write.**writerow**(stu)

csv\_write.**writerow**(stu1)

**print**("Write File Over!")

xml文件处理

**什么是xml文件？**

xml即**可扩展标记语言**，它可以用来标记数据、定义数据类型，是一种允许用户对自己的标记语言进行定义的源语言。

从结构上，很像HTML超文本标记语言。但他们被设计的目的是不同的，具体如下：

* XML 被设计用来传输和存储数据。
* HTML 被设计用来显示数据。

**<?xml version="1.0" encoding="utf-8"?>**

<note>

<to id='001'>George</to>

<from>John</from>

<heading>Reminder</heading>

<body>Don't forget the meeting!</body>

</note>

**xml特征：**

* 它是有标签对组成，<aa></aa>
* 标签可以有属性：<aa id='123'></aa>
* 标签对可以嵌入数据：<aa>abc</aa>
* 标签可以嵌入子标签（具有层级关系）

**XMl文件结构**

* XML 文档形成了一种树结构，它从“根部”开始，然后扩展到“枝叶”。
* 第一行是 XML 声明。它定义 XML 的版本 (1.0) 和所使用的编码
* <note>是根元素,也称为根节点。
* <to><from><heading><body>是子元素（子节点）
* XML 文档必须包含根元素。该元素是所有其他元素的父元素

**DOM文档对象模型**

**文档对象模型**（Document Object Model，简称DOM），DOM 就是针对 HTML 和 XML 提供的一个API。什么意思？就是说为了能以编程的方法操作这个 HTML 的内容（比如添加某些元素、修改元素的内容、删除某些元素），我们把这个 HTML或xml 看做一个对象树（DOM树），它本身和里面的所有东西比如 <div></div> 这些标签都看做一个对象，每个对象都叫做一个节点（node）。

**DOM 有什么用？**

就是为了操作 HTML或xml 中的元素，比如说我们要通过 JS 把这个网页的标题改了，直接这样就可以了：

document.title = '51zxw';

### 创建xml文件

创建一个xml文件Class\_info.xml 用来存储班级学生（姓名，年龄，城市），老师（姓名，年龄，城市）、教务账号（学生和老师的账号，密码）等信息。

**<?xml version="1.0" encoding="UTF-8" ?>**

<Class>

<student>

<name >Jack</name>

<age>28</age>

<city>Beijing</city>

</student>

<student>

<name >Bob</name>

<age>25</age>

<city>Shanghai</city>

</student>

<student>

<name>Harry</name>

<age>23</age>

<city>ShenZhen</city>

</student>

<teacher>

<name>Marry</name>

<age>23</age>

<city>Changsha</city>

</teacher>

<account>

<login username="student" password="123456"/>

<login username="teacher" password="888888"/>

</account>

</Class>

## xml节点

xml文件节点一般包含3类：

1. 元素节点
2. 文本节点
3. 属性节点

每个节点都拥有包含着关于节点某些信息的属性。这些属性是：

* nodeName（节点名称）
* nodeValue（节点值）
* nodeType（节点类型）

### 读取元素节点

### 案例：查看Class\_info.xml文件里Class节点的属性（结点名称，节点的值、节点类型）

from xml.dom import minidom

#加载xml文件

dom=minidom.**parse**('Class\_info.xml')

#加载dom对象元素

root=dom.documentElement

#打印节点信息

**print**(root.nodeName)

**print**(root.nodeValue)

**print**(root.nodeType)

* nodeName 节点名称
* nodeValue 返回文本节点的值
* nodeType 属性返回以数字值返回指定节点的节点类型。

1. 如果节点是元素节点，则 nodeType 属性将返回 1。
2. 如果节点是属性节点，则 nodeType 属性将返回 2。

### 读取文本节点的值

案例：分别打印出Class\_info.xml里的学生和老师的详细信息（姓名，年龄、城市）

from xml.dom import minidom

#获取标签对的值

#打开文件

dom=minidom.**parse**('Class\_info.xml')

#获取文档对象元素

root=dom.documentElement

#根据标签名称获取标签对象

names=root.**getElementsByTagName**('name')

ages=root.**getElementsByTagName**('age')

citys=root.**getElementsByTagName**('city')

#分别打印显示xml文档标签对里面的内容

**for** i **in** **range**(4):

**print**(names[i].firstChild.data)

**print**(ages[i].firstChild.data)

**print**(citys[i].firstChild.data)

### 读取属性节点的值

案例：分别读取打印老师和学生的账号密码。

from xml.dom import minidom

dom=minidom.**parse**('Class\_info.xml')

root=dom.documentElement

logins=root.**getElementsByTagName**('login')

#获取login标签的username属性

**for** i **in** **range**(2):

username=logins[i].**getAttribute**('username')

**print**(username)

password=logins[i].**getAttribute**('password')

**print**(password)

### 读取子节点信息

读取子节点<student>相关属性

* nodeName（节点名称）
* nodeValue（节点值）
* nodeType（节点类型）

from xml.dom import minidom

#加载xml文件

dom=minidom.**parse**('Class\_info.xml')

root=dom.documentElement

tags=root.**getElementsByTagName**('student')

**print**(tags[0].nodeName)

**print**(tags[0].tagName)

**print**(tags[0].nodeType)

**print**(tags[0].nodeValue)

多线程与多进程

#### 进程（Process）

是计算机中的程序关于某数据集合上的一次运行活动，是系统进行资源分配和调度的基本单位，是操作系统结构的基础。

#### 线程（Thread）

有时被称为轻量级进程(Lightweight Process，LWP），是程序执行流的最小单元。 线程是进程中的一个实体，是被系统独立调度和分派的基本单位，**一个进程可以包含多个线程，但是线程不能包含多个进程。**线程自己不拥有系统资源 ，在单个程序中同时运行多个线程完成不同的工作，称为多线程。

#### 线程与进程的区别

线程和进程的区别在于，子进程和父进程有不同的代码和数据空间，而多个线程则共享数据空间，每个线程有自己的执行堆栈和程序计数器为其执行上下文。

#### Tips：

LoadRunner和Jmeter性能测试工具也利用了**多线程和多进程**来构造多个并发用户来执行性能测试。

线程与进程图文解释

<http://www.ruanyifeng.com/blog/2013/04/processes_and_threads.html>

#### 单线程

单线程在程序执行时，所走的程序路径按照连续顺序排下来，前面的必须处理好，后面的才会执行。

案例：一个学生先用2秒说话，接着用3秒写字，最后结束。

**from** time **import** ctime,sleep

**def** **talk**():

print("Start talk %r" %ctime())

sleep(2)

**def** **write**():

print("Start Write! %r" %ctime())

sleep(3)

**if** \_\_name\_\_=="\_\_main\_\_":

talk()

write()

print("All end %r" %ctime())

if \_\_**name\_\_**=="\_\_**main\_\_**": 表示如果当前模块是被直接运行的，则该语句之后代码块被运行，如果模块是被导入的，则代码块不被运行。

#### 多线程

多线程（MultiThreading）是指从软件或者硬件上实现多个线程并发执行的技术。

案例：让学生同时进行说和写操作

from time import ctime,sleep

import threading

def **talk**(content,loop):

**for** i **in** **range**(loop):

**print**("Start Talk %s %s" %(content,ctime()))

**sleep**(3)

def **write**(content,loop):

**for** i **in** **range**(loop):

**print**("Start Write %s %s" %(content,ctime()))

**sleep**(5)

threads=[]

t1=threading.**Thread**(target=talk,args=('Speak: Hello,51zxw',2))

threads.**append**(t1)

t2=threading.**Thread**(target=write,args=('Write: Life is Short You need Python!',2))

threads.**append**(t2)

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

**for** t **in** threads:

t.**start**()

**for** t **in** threads:

t.**join**()

**print**("All the End %r" %ctime())

### 多进程

与多线程相比，多进程就是import multiprocessing 然后替换相应的方法multiprocessing.Process（）

from time import sleep,ctime

import multiprocessing

def **talk**(content,loop):

**for** i **in** **range**(loop):

**print**("Talk: %s %s" %(content,ctime()))

**sleep**(2)

def **write**(content,loop):

**for** i **in** **range**(loop):

**print**("Write: %s %s"%(content,ctime()))

**sleep**(3)

process=[]

p1=multiprocessing.**Process**(target=talk,args=('hello 51zxw',2))

process.**append**(p1)

p2=multiprocessing.**Process**(target=write,args=('Python',2))

process.**append**(p2)

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

**for** p **in** process:

p.**start**()

**for** p **in** process:

p.**join**()

**print**("All process is Run %s" %ctime())

#### 相关延伸：

* 进程间通信IPC(Interprocess communication)
* 线程锁，进程锁
* 生命周期
* 进程调度

Python 爬虫——爬取Web页面图片

从网页页面上批量下载jpg格式图片，并按照数字递增命名保存到指定的文件夹。

Web地址：<http://p.weather.com.cn/2017/06/2720826.shtml#p=1>

**import** urllib

**import** urllib.request

**import** re *#正则表达式*

*#解析页面*

**def** **load\_page**(url):

request=urllib.request.Request(url) *#发送网络请求*

response=urllib.request.urlopen(request)*#根据url打开页面*

data=response.read() *#获取页面响应数据*

**return** data

*#下载图片*

**def** **get\_image**(html):

regx=r'http://[\S]\*jpg' *#定义正则表达式，匹配页面图片元素*

pattern=re.compile(regx) *#编译表达式构造匹配模式*

get\_image=re.findall(pattern,repr(html)) *#进行正则匹配并返回结果*

num = 1

*#遍历获取的图片*

**for** img **in** get\_image:

image=load\_page(img)

*#将图片存入到指定文件夹*

**with** open('E:\\Photo\\%s.jpg' %num,'wb') **as** fb:

fb.write(image)

print("正在下载第 %s张图片" %num)

num = num + 1

print("下载完成！")

url='http://p.weather.com.cn/2017/06/2720826.shtml#p=1'

html=load\_page(url)

get\_image(html)

正则表达式相关知识：

<https://deerchao.net/tutorials/regex/regex.htm>