Python总结

python数据基本类型

- 1.不可变数据类型
  - 1.number。数字。
    - int,float,bool,complex
  - 2.string
  - 3.tuple
- 2.可变数据类型
  - 1.list
  - 2.dict
  - 3.set

Python的函数参数传递

- 1.当传入参数是不可变数据类型对象，则是值传递。
  - 不可变数据类型，不会影响到函数外的值。
- 2.当传入参数是可变数据类型对象，则是引用传递。
  - 可变数据类型，函数对其修改会影响到函数外的值。

Python 可哈希性和可变性

- 不可变的数据类型都是可哈希的。
- 可变的数据类型不是可哈希的。
  - 因为可变，所以程序运行时候其哈希值可能改变，所以存储位置也会发生改变，效率大打折扣。

python与c/c++混合编程

- 使用ctypes模块。
  - python的ctypes先将 C 编译成动态链接库的形式，即 Windows 下的 .dll 文件，或者 Linux 下的 .so 文件。

函数名后加箭头->,参数名后加冒号:

```
def func(a,text:str):->str
        pass
```

- 参数名后加冒号，说明参数名的类型。
  - text:str。说明参数text是str类型的。
- 函数名后加箭头->，对函数返回类型的注释。
  - ->。说明函数返回类型是str类型的。
- 注释信息保存再f.__annotations__中。
- python不做任何检查，不做强制，不做验证，所以这些信息仅仅是注释而已。

@staticmethod和@classmethod

- 1.静态方法(staticmethod)。
  - 静态方法调用类的属性或者方法时，只能通过类名.属性名/类名.方法名。
- 2.类方法(classmethod)。
  - 类方法函数的第一个参数必须是cls。
  - 类方法可以来调用类的属性，类的方法，实例化对象等，cls.func，避免硬编码。
- 3.实例方法(不带任何装饰器)。
  - 实例方法函数的第一个参数必须是self。
- 4.property
  - property() 函数的作用是在新式类中返回属性值。
  - @property装饰的函数被用做属性名，获取属性值的方法。
    - 如果属性是只读属性，若仍设置属性，则抛出异常。
  - @属性名.setter装饰的设置属性值的方法。
  - @属性名.deleter装饰的删除属性值的方法。

Python中的作用域

- 本地作用域（Local）→当前作用域被嵌入的本地作用域（Enclosing locals）→全局/模块作用域（Global）→内置作用域（Built-in）。

类变量和实例变量

- 1.类变量是可在类的所有实例之间共享的值。
- 2.实例变量是实例化之后，每个实例单独拥有的变量。

新式类和旧式类

- 新式类(python3默认)继承于object，旧式类(python2默认)不继承于object。
- 新式类的实例执行a.__class__与type(a)的结果是一致的，对于旧式类来说就不一样了。
- 在多继承中，新式类采用广度优先搜索，而旧式类是采用深度优先搜索。

如何在一个函数内部修改全局变量？

- 在函数内使用了global修改全局变量

***args,**kwargs什么意思？**

- 1.*args：允许将不定个数的变量作为参数传入函数。
- 2.**kwargs：允许将不定个数的键值对作为参数传入函数。

__init__和__new__区别

- 1.__new__是一个静态方法,而__init__是一个实例方法。
- 2.__new__方法会返回一个创建的实例,而__init__什么都不返回。
- 3.只有在__new__返回一个class的实例时后面的__init__才能被调用。
  - __new__()在__init__()之前被调用，用于生成实例对象。
- 4.当创建一个新实例时调用__new__,初始化一个实例时用__init__。

避免转义给字符串加哪个字母表示原始字符串？

- 字符串前加r,说明之后的字符串是原始字符串。

type和**isinstance**异同

- 共同点
  - type和isinstance都可以判断变量是否属于某个内建类型。
- 不同点
  - type只接收一个参数，不但可以判断变量是否属于某个类型，而且可以得到参数变量未知的所属的类型。isinstance只能判断是否属于某个已知类型，不能直接得到变量未知的所属的类型
  - isinstance可以判断子类实例对象是属于父类的(子类和父类类型一致)；而type会判断子类实例对象和父类类型不一样。

is和**==**区别

- Python中对象包含的三个基本要素
  - id(身份标识)
  - type(数据类型)
  - value(值)
- 1.is
  - is比较的是两个对象的id值是否相等，也就是比较是否指向同一个内存地址.
- 2.==
  - ==比较的是两个对象的内容(value)是否相等，默认会调用对象的__eq__()方法。

装饰器：

- 装饰器就是一个接受函数为参数，并完成一些操作的函数。
- 使用@声明一个函数是装饰器。
- 使用@语法后，在创建被装饰的可调用函数后，会立刻应用装饰器，不用得到main调用到该装饰后的函数。
- 一个函数存在多个装饰器时，自底向上的顺序来应用装饰器。

上下文管理器：

- 上下文管理器保证进入上下文管理器时，每次执行代码的一致性。退出上下文管理器时，相关资源会被正确回收。
- 使用with语句进入上下文管理器，并将__enter__的返回值赋给as之后的变量。
- 定义一个上下文管理器必须定义一个__enter__方法和__exit__方法。__enter__除了self，不接受其他参数，__exit__除了self，还有3个位置参数(异常类型，异常实例，回溯)。或者使用装饰器*
- @contextlibr.contextmanager定义一个上下文管理器。

生成器：

- 生成器是一个函数，并按照一定顺序返回一个或者多个值，而不是返回一个单一值。
- 使用yield。yield返回一个值，但是不会终止函数的运行，而是暂停直到调用函数重新调用生成器。
- next请求生成器的下一个值。
- Python2中生成器中不能有return(即yield和return不能共存)，通过raise StopIteration相当于return。

- Python3中yield和return可以共存，但是return返回的值会被当成异常信息发送。return 42 相当于raise StopIteration(42)。
- send可以赋给yield表达式的值。
- 生成器内部的生成器。yield from 函数。

迭代器：

- 生成器是一种迭代器。
- Python中的迭代器是包含__next__方法的任何对象。
- Python中的迭代对象是任何定义了__iter__方法的对象。迭代对象的__iter__方法返回一个迭代器。
- 生成器：iter(range)，iter(dict.item)，zip，map和文件对象。

魔术方法：

- 魔术方法被用于重载python的操作符和内置方法。
- 命名方式：将下划线放到方法名称的两端（__init__）。
- 当特定语法出现时，魔术方法作为执行的钩子（钩子就是特定事件发生时，能够为响应事件而调用的代码或函数）。

Python里和None比较时，为什么是 is None 而不是 == None 呢？

- 因为None在Python里是个单例对象，一个变量如果是None，它一定和None指向同一个内存地址。

元类：

- 元类(Metaclass)是生成其他类的类。
- 道(type)生一(metaclass)，一生二(class)，二生三(instance)，三生万物。
- type是元类层级的最高级。Object是类继承链中的最高级。
- 元类是type的子类。任何类都只能有一个元类。只有一个元类是另一个元类的直接子类时，才能接受一个类继承自使用不同子类的两个类。
- __new__(name(类名),base(基类),attrs(字典))。编写元类要求重载__new__方法，但是通常不用实现__init__方法。
- 可以通过直接调用Meta元类来创建使用Meta元类的类。

类工厂：

- 类工厂时一个在运行时创建类的函数。
- 类工厂函数就是一个用于创建并返回类的函数。
- 编写类工厂函数可以在需要基于运行时的信息而创建类。每次返回都不是并不是同一个类。

抽象基类：

- 规定了继承类必须具有抽象基类指定的方法，抽象基类无法实例化。 任何抽象基类都必须使用ABCMeta元类。
- ABCMeta的实例通过使用register方法提供了对声明的实现。register也可以作为装饰器。
- __subclassblock__声明必须被子类实现的方法，优先级高于register。
- 任何带有需实现方法的类都被认为是抽象基类的子类，无需有继承关系。
    - @abstractmethod：抽象方法，必须被子类重写的方法。

    - @property：抽象属性。与@abstractmethod协作。
- @staticmethod，@classmethod：静态方法，属于类，无须创建对象就可以调用。@classmethod第一个参数是表示自身类的cls, @staticmethod无须参数。

匿名函数

```
lambda arg1,arg2,.....argn:expression

def fun(arg1,arg2,....):
        return expression
```

cls和self区别

- cls
    - cls是类方法的参数，表示这个类的本身。
- self
    - self是普通方法的参数，表示一个具体的实例本身。
    - self是__new__的返回值。

Python函数的默认参数

- 默认参数语句，总是在 def 关键字定义函数的时候被求值，且仅执行这一次。
- 默认参数只会运算一次，而不是每次被调用时都会重新运算。

Python管理内存

- Python 有一个私有堆空间来保存所有的对象和数据结构。
- Python有内存池机制和Pymalloc机制。
    - 内存池预先在内存中申请一定数量的，大小相等的内存块留作备用，当有新的内存需求时，就先从内存池中分配内存给这个需求，不够了之后再申请新的内存。
    - 当申请空间少于256KB时在内存池中申请空间，否则直接malloc来申请内存空间。
    - Python对象(float,int,list...)都有独立的私有内存池，对象间不分享内存池。

help() 函数

- help函数返回帮助文档和参数说明。

dir() 函数

- dir函数返回对象中所有的成员。

当退出 Python 时是否释放所有内存分配？

- 否。
    - 具有对象循环引用或者全局命名空间引用的变量，在 Python 退出是往往不会被释放。
    - 不会释放 C 库保留的部分内容。

推导式

- 推导式是可以从一个数据序列构建另一个新的数据序列的结构体。
- 1.列表推导式
    - 1.使用[]生成list

```
my_list = [out_exp_res for out_exp in input_list if condition]
```

    - 2.使用()得到生成器

```
multiples = (i for i in range(30) if i%3 == 0)
```

- 2.字典推导式

```
my_dict={ key_expr: value_expr for value in collection if condition }
```

- 3.集合推导式

```
my_set={ expr for value in collection if condition }
```

三元运算符

```
condition_is_true if condition else condition_is_false

(if_test_is_false, if_test_is_true)[test]
```

猴子补丁

- 猴子补丁的意思是在程序运行时(runtime)修改某些代码。

map,filter和reduce

- map将一个函数映射到一个输入列表的所有元素上。

```
map(function_to_apply, list_of_inputs)
```

- filter过滤列表中的元素，并且返回一个由所有符合要求的元素所构成的列表。

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

- reduce函数会对参数序列中元素进行累积。
    - reduce 中的函数 function（假设有两个参数）先对集合中的第 1、2 个元素进行操作，得到的结果再与第三个数据用 function 函数运算，最后得到一个结果。

      reduce(function, iterable[, initializer])

对象变动**(mutation)**

- 1.当你将一个变量赋值为另一个可变类型的变量时，对这个数据的任意改动会同时反映到这两个变量上去。新变量只不过是老变量的一个别名而已。

__slot__魔法

- 1.告诉Python不要使用字典，而且只给一个固定集合的属性分配空间。

@property

- @property 把方法『变成』了属性。

调用父类同名方法

- 调用未绑定的父类方法。
    - Base.func()
    - 菱形继承会出现多次初始化基类。
- 使用super调用父类方法。
    - MRO列表(Method Resolution Order)(解决菱形继承问题)
        - 子类永远在父类前面
        - 如果有多个父类，会根据它们在列表中的顺序被检查
        - 如果对下一个类存在两个合法的选择，选择第一个父类
    - super 和父类没有实质性的关联。
    - super(cls, inst) 获得的是 cls 在 inst 的 MRO 列表中的下一个类。

容器**(collections)**

- 1.defaultfict
    - 与dict类型不同，你不需要检查key是否存在。

闭包

- 作用：保存局部信息不被销毁，保存当前的运行环境。
- 概念：在一个内部函数中，对外部作用域的变量进行引用，(并且一般外部函数的返回值为内部函数)，那么内部函数就被认为是闭包。
    - 1.必须有一个内嵌函数。

    - 2.内嵌函数必须引用外部函数中的变量。
    - 3.外部函数的返回值必须是内嵌函数。

```
def startAt(X):
    def incrementBy(y):    *incrementBy是闭包
        return x+y
    return incrementBy
```

- 闭包无法修改外部函数的局部变量

i = i+1 和 i += 1

- 对于可变类型对象 i+=1和i= i+1都会改变内存地址。
- 对于可变类型对象 i+=1不会改变内存地址, i = i+1会改变内存地址。

join和+拼接字符串的区别

- join的性能好于+。
- string是不可变对象。
- 使用+来拼接字符串时，每次申请一块新的内存空间，将结果放入新的内存空间中。
- 使用join来拼接字符串时，会先计算结果所需的内存，然后一次性申请所需内存。

append和extend的区别

- 1.append向列表list中添加一个对象object。
    - 添加元素当成一个整体加入到list中。
- 2.extend把一个序列sequence的内容添加到列表list中。
    - 添加元素逐个加入到list中。

下划线

- 1.单前导下划线：_var
    - 命名约定，仅供内部使用。
    - 私有变量，但是又可以外部访问。
        - Python中的单个下划线前缀仅仅是一个约定
    - 如果使用通配符(*)从模块中导入所有名称，则Python不会导入带有前导下划线的名称。常规导入不受前导单个下划线命名约定的影响。
        - 除非模块定义了覆盖此行为的__all__列表。
- 2.单末尾下划线：var_
    - 单个末尾下划线（后缀）是一个约定，用来避免与Python关键字产生命名冲突。
- 3.双前导下划线：__var
    - 当在类上下文中使用时，触发名称修饰。
        - 双下划线前缀会导致Python解释器重写属性名称，以避免子类中的命名冲突。
- 4.双前导和末尾下划线：__var__
    - 由双下划线前缀和后缀包围的变量不会被Python解释器修改。
    - 魔术方法。

- 单下划线：_
    - 用作临时或无意义变量的名称。

docstring

- 是为函数、模块和类注释生成文档。

read，readline，，readlines和xreadlines

- read()会读取整个文件，将读取到底的文件内容放到一个字符串变量，返回str类型。
- readline()读取一行内容，放到一个字符串变量，返回str类型。
- readlines() 读取文件所有内容，按行为单位放到一个列表中，返回list类型。
- xreadlines()返回一个生成器，来循环操作文件的每一行。

垃圾回收机制

- python采用的是引用计数机制为主，标记-清除和分代回收（隔代回收）两种机制为辅的策略。
- 1.引用计数
- 2.分代回收
    - 分代回收是一种以空间换时间的操作方式，Python将内存根据对象的存活时间划分为不同的集合，每个集合称为一个代，Python将内存分为了3“代”，分别为年轻代（第0代）、中年代（第1代）、老年代（第2代），他们对应的是3个链表，它们的垃圾收集频率与对象的存活时间的增大而减小。
    - 新创建的对象都会分配在年轻代，年轻代链表的总数达到上限时，Python垃圾收集机制就会被触发，把那些可以被回收的对象回收掉，而那些不会回收的对象就会被移到中年代去，依此类推，老年代中的对象是存活时间最久的对象，甚至是存活于整个系统的生命周期内。
- 3.标记清除（Mark—Sweep）
    - 标记清除是基于追踪回收（tracing GC）技术实现的垃圾回收算法。
        - 第一阶段是标记阶段，GC会把所有的『活动对象』打上标记。
        - 第二阶段是把那些没有标记的对象『非活动对象』进行回收。
        - 对象之间通过引用（指针）连在一起，构成一个有向图，对象构成这个有向图的节点，而引用关系构成这个有向图的边。
        - 从根对象（root object）出发，沿着有向边遍历对象，可达的（reachable）对象标记为活动对象，不可达的对象就是要被清除的非活动对象。
        - 根对象就是全局变量、调用栈、寄存器。
        - 清除非活动的对象前它必须顺序扫描整个堆内存，哪怕只剩下小部分活动对象也要扫描所有对象。
- 垃圾回收器会定时寻找引用循环，并将其回收。

内存分配机制

- 1.大于256k的内存使用malloc分配。
- 2.小于等于256k的内存使用内存池分配。
    - 第-1，-2层：操作系统进行操作。
    - 第0层：大内存-----若请求分配的内存大于256K，malloc函数分配内存，free函数释放内存。
    - 第1层和第2层：内存池，有Python的接口函数PyMem_Malloc实现-----若请求分配的内存在1~256KB之间就使用内存池管理系统进行分配，调用malloc函数分配内存，但是每次只会分配一

块大小为256K的大块内存，不会调用free函数释放内存，将该内存块留在内存池中以便下次使用。
    - 第3层：最上层，用户对Python对象的直接操作。
    - 经由内存池登记的内存到最后还是会回收到内存池，并不会调用C的free释放掉，以便下次使用。

python异常处理机制

- 1.捕捉异常可以使用try/except语句。
    - 当开始一个try语句后，python就在当前程序的上下文中作标记，这样当异常出现时就可以回到这里，try子句先执行
        - 1.如果当try后的语句执行时发生异常，python就跳回到try并执行第一个匹配该异常的except子句，异常处理完毕，控制流就通过整个try语句（除非在处理异常时又引发新的异常）。
        - 2.如果在try后的语句里发生了异常，却没有匹配的except子句，异常将被递交到上层的try，或者到程序的最上层（这样将结束程序，并打印缺省的出错信息）。
        - 3.如果在try子句执行时没有发生异常，python将执行else语句后的语句（如果有else的话），然后控制流通过整个try语句。
- 2.final语句
    - finally 语句无论是否发生异常都将执行。

Exception 和 BaseException

- 1.BaseException是最基础的异常类，Exception继承了它。
- 2.捕获所有异常时更应该使用Exception，更高级别的异常交给Python解释器处理。

GIL(Global Interpreter Lock)锁

- 1.GIL不是python的特性，而是python解释器(CPython)所引入的概念。
- 2.GIL是全局排他锁，只要申请到锁的线程才可以获得cpu。即使多核，同一个时刻也只有一个线程运行。

__import__和import区别

- import是导入/引入一个python标准模块，其中包括.py文件、带有__init__.py(python3中有无无所谓)文件的目录。
- __import__是一个函数，功能与import一致，将需导入的包当成参数传入到__import__中。

import机制

- 模块和包
    - 1.模块就是一组功能的组合，可以看成是一个.py文件。
    - 2.包就是一个包含python模块的文件夹,并包含__init__文件。
        - 使用from pacakge import *，需要在包的__init__.py文件里加上:__all__=['a.py','b.py',...]。
        - __path__默认只有一个元素(当前包的路径)。
- sys.module是将模块名称映射到已加载的模块字典里。
- 命名空间

    - 局部命名空间(local namespace)
        - 每个函数的命名空间，记录该函数的变量。
    - 全局命名空间(global namespace)
        - 每个模块的命名空间，记录该模块的变量。
    - build-in命名空间(build_in namespace)
        - 包含build-in function和exceptions，可以被任意模块访问。
- import机制
    - 标准import导入
        - 1.import module_name。
        - 2.module_name是否在sys.modules字典中。
            - 在sys.modules字典中，则将module_name加入到当前文件的local命名空间中。
            - 不在sys.modules字典中，则从sys.path目录中查找module_name文件，并将其载入内存，加入到sys.modules字典中，最后将module_name加入到当前文件的local命名空间中。
    - 嵌套import导入
        - 顺序import导入
            - import module_a
            - module_a文件中import module_b....
        - 循环/嵌套导入
            - 循环导入时，两个.py文件的global命令空间都为空，访问变量会抛出异常。
            - 解决方法
                - 延迟导入。把import语句写在方法/函数里，将import作用域限制在局部。
                - 将from xx import yy 改成 import xx.yy形式
                - 组织代码（重构代码）。更改代码布局，可合并或分离竞争资源。
                    - 合并。都写在一个.py文件里。
                    - 分离。把需要import的资源都提取到第三方的.py文件中。
    - 包（package）import
        - 包的导入和模块导入基本一致，只是导入包时，会执行这个 __init__.py，而不是模块中的语句。
            - import xxx时，而包的__init__.py文件没有初始化，则包下的模块不会自动导入。

python2与python3区别

- print
    - python2中print是一个类。
    - python3中print是一个函数。必须使用括号。
- input
    - python2通过input得到int类型，使用raw_input得到str类型。
    - python3通过input得到str类型。
- range
    - python2中xrange返回一个迭代器，range返回一个列表。
    - python3中range返回一个迭代器。
- /
    - python2中/结果类型取决于两个运算数,若两个数中有一个是float，则返回float，否则返回int。
    - python3中返回int。
- import机制
    - python2采用相对路径进行import。

    - python3在需要导入同一目录的文件必须使用绝对路径，否则只能使用相关导入的方式来进行导入。
- 编码方式
    - python2文件默认编码是ASCII，字符串默认也是ACSII。
    - python3文件默认编码是utf-8，字符串默认是unicode。即使声明了某种编码，在内存里还是unicode。
- 缩进机制
    - python2缩进机制中，允许同时使用tab和space，1 tab==8 space。
    - python3缩进机制中，不需要tab和space共存。
- for循环
    - Python2中for 循环会修改外部相同名称变量的值。
    - Python3中for 循环不会修改外部相同名称变量的值
- 比较操作符
    - Python2 中任意两个对象都可以比较。
    - Python3 中只有同一数据类型的对象可以比较。否则会抛出TypeError。
- 不等运算符
    - Python2中不等于有两种写法 != 和 <>。
    - Python3中去掉了<>, 只有!=一种写法。
- 类
    - python2默认是老式类，除非继承object。
    - python3默认是新式类，默认继承object。
- 异常处理

    - python2

```
except Exception, e:
       .......
```

    - python3

```
except Exception as e:
       ......
```

Flask总结

Flask和Django

- 1.Flask是轻量级web框架,默认依赖两个外部库：jinja2和Werkzeug WSGI工具。
- 2.Django是重量级web框架,功能齐全。
    - 自带ORM(Object-Relational Mapping 对象关系映射)和模板引擎,支持jinja等非官方模板引擎。
    - 自带数据库管理app。

Flask框架依赖组件

- 1.Route(路由)。
- 2.templates(模板)。
- 3.Models(orm模型)。
- 4.blueprint(蓝图)。
    - 蓝图Blueprint实现模块化的应用。
    - 将不同的功能模块化，优化项目结构，增强可读性,易于维护。
- 5.Jinja2模板引擎。

WSGI

- WSGI（Web Server Gateway Interface,Web 服务器网关接口）则是Python语言中所定义的Web服务器和Web应用程序之间或框架之间的通用接口标准。
    - WSGI就是一座桥梁,桥梁的一端称为服务端或网关端,另一端称为应用端或者框架端,WSGI的作用就是在协议之间进行转化。
- WSGI将Web组件分成了三类：Web 服务器（WSGI Server）、Web中间件（WSGI Middleware）与Web应用程序（WSGI Application）。
- Web Server接收HTTP请求,封装一系列环境变量,按照WSGI接口标准调用注册的WSGI Application,最后将响应返回给客户端.

如何在**Flask**中访问会话？

wtforms组件的作用

- WTForms是一个支持多个web框架的form组件，主要用于对用户请求数据进行验证。

Flask上下文管理流程

- 每次有请求过来的时候，flask 会先创建当前线程或者进程需要处理的两个重要上下文对象，把它们保存到隔离的栈里面，这样视图函数进行处理的时候就能直接从栈上获取这些信息。

Flask框架默认**session**处理机制

- Flask的默认session利用了Werkzeug的SecureCookie,把信息做序列化(pickle)后编码(base64),放到cookie里了。
- 过期时间是通过cookie的过期时间实现的。
- 为了防止cookie内容被篡改,session会自动打上一个叫session的hash串。
    - hash串是经过session内容、SECRET_KEY计算出来的。

Flask-WTF

- Flask-wtf是一个用于表单处理,校验并提供csrf验证的功能的扩展库。
- Flask-wtf能把正表单免受CSRF<跨站请求伪造>的攻击。

Flask中的数据库连接

- 1.在脚本中以用第三方库正常连接,用sql语句正常操作数据库。
- 2.用ORM来进行数据库连接。flask_sqlalchemy。

ORM的实现原理