目录

1.标准数据类型？

Numbers（数字）
String（字符串）
List（列表）：有序的对象集合
Tuple（元组）：元组用()标识,内部元素用逗号隔开。但是元组不能二次赋值，相当于只读列表。
Set（集合）
Dictionary（字典）:无序的对象集合
Python3 支持 int、float、bool、complex（复数）；没有 Long。

不可变数据：**Number**（数字）.**String**（字符串）.**Tuple**（元组）；
不允许变量的值发生变化，如果改变了变量的值，相当于是新建了一个对象，而对于相同的值的对象，在内存中则只有一个对象（一个地址）
可变数据：**List**（列表）.**Dictionary**（字典）.**Set**（集合）。
允许变量的值发生变化，即如果对变量进行 append.+=等这种操作后，只是改变了变量的值，而不会新建一个对象，变量引用的对象的地址也不会变化;
每个对象都有自己的地址

2.如何创建一个字典？

2.1 传入关键字 dict(a='a', b='b', t='t')
2.2 映射函数方式来构造字典
dict(zip(['one', 'two', 'three'], [1, 2, 3]))
2.3 可迭代对象方式来构造字典
dict([('one', 1), ('two', 2), ('three', 3)])

3.双下划线和单下划线的区别？

3.1 前后都有双下划线-特殊变量
__xxx__: 是可以直接访问的。
3.2 前面双下划线-私有变量
__xxx: 只有内部可以访问，外部不能访问;但是可以通过_${classname}__name 来访问。
3.3 前面单下划线-口头私有变量
_xxx: 外部是可以访问的;但是请把我视为私有变量，不要随意访问。

4.自省解释一下？

运行时能够获得对象的类型：type(),dir(),getattr(),hasattr(),isinstance().

5.文件可以使用 for 循环进行遍历？ 文件对象实现了迭代器协议

6.迭代器和生成器的区别

	生成器	迭代器
联系	生成器是一种特殊的迭代器	
区别	通过函数的形式中调用 **yield** 或()的形式创建的	通过 **iter**()内置函数创建
	调用next()函数或for循环中，所有过程被执行，且返回值；只能遍历一次；好处是**延迟计算**，一次返回一个结果。它不会一次生成所有的结果，对于大数据量处理有利	调用next()函数或for循环中，所有值被返回，没有其他过程或动作

7.*args and **kwargs

*args:不确定函数里要传递多少参数;
**kwargs:允许使用没有事先定义的参数名.

8.装饰器怎么用？装饰器解释下，基本要求是什么？

装饰器的作用就是为已经存在的对象添加额外的功能;
参数为函数，返回为函数，本质是嵌套函数.
装饰器可以传参，也可以不用传参。
内置装饰器有三种: **@property、@staticmethod、@classmethod**
@property: 把类内方法当成属性来使用，必须要有返回值，相当于 getter.

9.新式类和旧式类区别

9.1 新式类都从 object 继承，经典类不需要。
9.2 新式类的 MRO(method resolution order 基类搜索顺序)算法采用 C3 算法广度优先搜索，而旧式类的 MRO 算法是采用深度优先搜索。
9.3 新式类相同父类只执行一次构造函数，经典类重复执行多次。

10.__new__和__init__的区别

__new__是一个静态方法,而__init__是一个实例方法.
__new__方法会返回一个创建的实例,而__init__什么都不返回.
只有在__new__返回一个 cls 的实例时后面的__init__才能被调用.
当创建一个新实例时调用__new__,初始化一个实例时用__init__.

11.单例模式的几种实现方式的及优化？

1.使用模块

```
# mysingleton.py
class My_Singleton(object):
    def foo(self):
        pass

my_singleton = My_Singleton()

# to use
from mysingleton import my_singleton

my_singleton.foo()
```

2.使用装饰器

```
def singleton(cls):
    _instance = {}

    def inner():
        if cls not in _instance:
            _instance[cls] = cls()
        return _instance[cls]
    return inner

@singleton
class Cls(object):
    def __init__(self):
        pass

cls1 = Cls()
cls2 = Cls()
print(id(cls1) == id(cls2)) #True
```

3. 使用类装饰器

```
class Singleton(object):
    def __init__(self, cls):
        self._cls = cls
        self._instance = {}
    def __call__(self):
        if self._cls not in self._instance:
            self._instance[self._cls] = self._cls()
        return self._instance[self._cls]

@Singleton
class Cls2(object):
    def __init__(self):
        pass

cls1 = Cls2()
cls2 = Cls2()
print(id(cls1) == id(cls2)) #True
```

4. 基于__new__方法实现（推荐使用，方便）

```
class Single(object):
    _instance = None
    def __new__(cls, *args, **kw):
        if cls._instance is None:
            cls._instance = object.__new__(cls, *args, **kw)
        return cls._instance
    def __init__(self):
        pass

single1 = Single()
single2 = Single()
print(id(single1) == id(single2)) #True
```

5.基于 metaclass 方式实现

```python
class Singleton(type):
    _instances = {}
    def __call__(cls, *args, **kwargs):
        if cls not in cls._instances:
            cls._instances[cls] =
                super(Singleton, cls).__call__(*args, **kwargs)
        return cls._instances[cls]

class Cls4(metaclass=Singleton):
    pass

cls1 = Cls4()
cls2 = Cls4()
print(id(cls1) == id(cls2)) #True
```

12.作用域的类型有哪些？

局部作用域： 在函数中定义的变量
嵌套作用域： 为了实现 Python 的闭包
全局作用域： 作用范围仅限于单个模块文件内
内置作用域： 系统内固定模块里定义的变量

搜索变量名的优先级：局部作用域 > 嵌套作用域 > 全局作用域 > 内置作用域

```python
def test_scopt():
    variable = 200
    print(variable)
    def func():
        print(variable)
    func()
variable = 100

test_scopt() # 200 200
print(variable) # 100
```

13.深拷贝和浅拷贝的区别？

深拷贝：对一个对象的拷贝做出改变时，不会影响原对象；使用：deepcopy
浅拷贝：在拷贝中改动，也会影响到原对象；使用：copy

14.多线程和多进程的区别？

多线程: threading 多进程:multiprocessing
1.多线程可以共享全局变量，多进程不能
2.多线程中，所有子线程的进程号相同；多进程中，不同的子进程进程号不同
3.线程共享内存空间；进程的内存是独立的
4.同一个进程的线程之间可以直接交流；两个进程想通信，必须通过一个中间代理来实现
5.创建新线程很简单；创建新进程需要对其父进程进行一次克隆
6.一个线程可以控制和操作同一进程里的其他线程；但是进程只能操作子进程
两者最大的不同在于：在多进程中，同一个变量，各自有一份拷贝存在于每个进程中，互不影响；而多线程中，所有变量都由所有线程共享。

```python
import threading
import time
def start(arg):
    time.sleep(0.5)
    print("%s running..... \n" % arg)

if __name__ == '__main__':
    t1 = threading.Thread(target=start,args=('This is thread 1',))
    t2 = threading.Thread(target=start,args=('This is thread 2',))
    t1.start()
    t2.start()
    print('This is main function')
```

```
This is main function
This is thread 1 running.....
This is thread 2 running.....
```

15.is 是对比地址,==是对比值

16. read,readline 和 readlines

read 读取整个文件
readline 读取下一行,使用生成器方法
readlines 读取整个文件到一个迭代器以供我们遍历

17.闭包

形成的条件:

1.外部函数中定义了内部函数
2.外部函数有返回值
3.返回的值是：内部函数名
4.内部函数引用了外部函数的变量

```python
# 外部函数
def outer_func(a, b):
    c = 10
    # 内部函数
    def inner_func():
        sum = a + b + c  # 调用了外部函数变量c
        print('总和为：{}'.format(sum))
    return inner_func  # 返回内部函数名

ifunc = outer_func(2, 3)  # 这里其实已将内部函数的地址已经给了ifunc
ifunc()  # 完了之后加上小括号进行函数调用
```

18.垃圾回收机制?

GC 主要使用引用计数来跟踪和回收垃圾。
在引用计数的基础上，通过"标记-清除”解决容器对象可能产生的循环引用问题，通过"分代回收”以空间换时间的方法提高垃圾回收效率。

19. +和 join 的区别？

“+”：可以连接多个字符串产生一个字符串对象；
join：将多个字符串采用固定的分隔符连接在一起。
join 只会进行一次内存申请，因此运行效率相对于+会快很多。
注： 合并 str1 和 str2，用 str1+str2

20.为什么要使用 Lambda 函数？怎么使用？

也称为丢弃函数，可以与其他预定义函数（filter(),map()等）一起使用。
相对于定义的可重复使用的函数来说，这个函数更加简单便捷。

```
from functools import reduce
my_list = [2,3,4,5,6,7,8]
new_list_filter = list(filter(lambda a:(a / 3 == 2),my_list))
new_list_map = list(map(lambda a:(a / 3!= 2),my_list))
new_list_reduce = reduce(lambda a,b: a+b,my_list)
print(new_list_filter)  # [6]
print(new_list_map)     # [True, True, True, True, False, True, True]
print(new_list_reduce)  # 求和 ： 35
```

21.协程的理解？怎么使用？

协程的作用：是在执行函数 A 时可以随时中断去执行函数 B，然后中断函数 B 继续执行函数 A（可以自由切换）。
但这一过程并不是函数调用，这一整个过程看似像多线程，然而协程只有一个线程执行。
python2.x 实现协程的方式有： yield + send gevent
Python3.x 协程：

asyncio + **yield from** (python3.4+) **asyncio** + **async/await** (python3.5+)

22.谈下 python 的 GIL?

GIL 是 python 的全局解释器锁，同一进程中假如有多个线程运行，一个线程在运行 python 程序的时候会霸占 python 解释器（加了一把锁即 GIL），使该进程内的其他线程无法运行，等该线程运行完后其他线程才能运行。如果线程运行过程中遇到耗时操作，则解释器锁解开，使其他线程运行。所以在多线程中，线程的运行仍是有先后顺序的，并不是同时进行。
多进程中因为每个进程都能被系统分配资源，相当于每个进程有了一个 python 解释器，所以多进程可以实现多个进程的同时运行，缺点是进程系统资源开销大

23.字典如何删除键和合并两个字典?

del 和 **update** 方法

24.列出 5 个 python 标准库?

os sys re math datetime

去首尾空格： a.strip()
去空格：replace 或 split

25.字典和 json 字符串相互转化方法？

json.dumps()：字典 转 json 字符串；
json.loads()：json 转 字典

26.列表去重的方法?

1.**set** 集合去重
2.**fromkeys**:用列表中的元素作为字典中的 key 生成一个新字典，然后获取字典的 key
3.引入 **defaultdict**
4.最简单的循环，添加入新的列表，如果新列表中没有相同元素，则加入
5.引入 itertools 的 **groupby** 方法
6.**reduce** 方法

27.python2 和 python3 的 range（100）的区别？

python2 返回列表，python3 返回迭代器，节约内存

28.python2 和 python3 区别？列举 5 个

1.Python3 使用 print 必须要以小括号包裹打印内容，比如 print('hi')
Python2 既可以使用带小括号的方式，也可以使用一个空格来分隔打印内容，比如 print 'hi'
2.python2 range(1,10)返回列表，python3 中返回迭代器，节约内存
3.python2 中使用 ascii 编码，python 中使用 utf-8 编码
4.python2 中 unicode 表示字符串序列，str 表示字节序列
python3 中 str 表示字符串序列，byte 表示字节序列
5.python2 中为正常显示中文，引入 coding 声明，python3 中不需要
6.python2 中是 raw_input()函数，python3 中是 input()函数

29.列出几种魔法方法并简要介绍用途？

__init__:对象初始化方法
__new__:创建对象时候执行的方法，单列模式会用到
__str__:当使用 print 输出对象的时候，只要自己定义了__str__(self)方法，那么就会打印从在这个方法中 return 的数据
__del__:删除对象执行的方法

30.异常的解释？

IOError：输入输出异常
AttributeError：试图访问一个对象没有的属性
ImportError：无法引入模块或包，基本是路径问题
IndentationError：语法错误，代码没有正确的对齐
IndexError：下标索引超出序列边界
KeyError:试图访问你字典里不存在的键
SyntaxError:Python 代码逻辑语法出错，不能执行
NameError:使用一个还未赋予对象的变量

31.sort 函数内部实现原理？

内部实现是 timsort,Timsort 是结合了合并排序和插入排序而得出的排序算法，它在现实中有很好的效率。
该算法找到数据中已经排好序的块-分区，每一个分区叫一个 run，然后按规则合并这些 run。
算法的过程包括：

1.如何数组长度小于某个值，直接用二分插入排序算法
2.找到各个 run，并入栈
3.按规则合并 run

如何提高 python 的运行效率?

1.使用生成器，因为可以节约大量内存
2.循环代码优化，避免过多重复代码的执行
3.核心模块用 Cython　PyPy 等，提高效率
4.多进程,多线程,协程
5.多个 if elif 条件判断，可以把最有可能先发生的条件放到前面写，这样可以减少程序判断的次数，提高效率

遇到 bug 如何处理?

1.细节上的错误，通过 print（）打印，能执行到 print（）说明一般上面的代码没有问题，分段检测程序是否有问题，如果是 js 的话可以 alert 或 console.log
2.如果涉及一些第三方框架，会去查官方文档或者一些技术博客。
3.对于 bug 的管理与归类总结，一般测试将测试出的 bug 用 teambin 等 bug 管理工具进行记录，然后我们会一条一条进行修改，修改的过程也是理解业务逻辑和提高自己编程逻辑缜密性的方法，我也都会收藏做一些笔记记录。

常用 Linux 命令?

ls,help,cd,more,clear,mkdir,pwd,rm,grep,find,mv,su,date