讲课内容:

1. 初识函数

2. 函数的返回值

3. 函数的传参(包含\*args,\*\*kwargs)

4. 函数的嵌套和作用域

5. 关键字global,nonlocal

6. 内置函数(部分)

**一. 初识函数**

**1.1. 什么是函数**

我们到目前为止, 已经可以完成一些软件的基础功能了. 那么我们来完成这样一个功能: 约x:

print("拿出手机")

print("打开陌陌")

print("找个漂亮的妹子")

print("问她, 约不约啊?")

print("oK. 走你!")

ok. so easy. 我们已经完成了对一个功能的描述. 那么问题来了. 我还想再约一次. 怎么办呢?  很简单. 再写一次就好了

# 约一次

print("拿出手机")

print("打开陌陌")

print("找个漂亮的妹子")

print("问她, 约不约啊?")

print("oK. 走你!")

# 再来一次

print("拿出手机")

print("打开陌陌")

print("找个漂亮的妹子")

print("问她, 约不约啊?")

print("oK. 走你!")

OK. 也很简单. 但是. 我现在还想约. 约个10次8次的. 怎么办呢? 也简单. 加个循环就好了

while 1:

    print("拿出手机")

    print("打开陌陌")

    print("找个漂亮的妹子")

    print("问她, 约不约啊?")

    print("oK. 走你!")

哇, 终于可以不停的约了.但是呢, 想想. 这样写出来的程序. 是不是一直在约? 人啊. 要有节制. 有需求了再约, 这样比较好. 所以呢.这样写是不行的. 最好是我想什么时候约就什么时候约. 好了. 说到这. 我们可以这样做, 把约会这个事情啊, 先计划一下, 然后呢安排好流程. 在需要约的时候呢. 把这个约的流程拿出来执行一下就好了. 那么这里.我们可以先去定义一个事情或者功能. 等到需要的时候直接去用就好了. 那么这里定义的东西就是一个函数.

函数: 对代码块和功能的封装和定义

我们使用def关键字来定义函数, 函数的定义语法:

def 函数名():

    函数体

这里的函数名的命名规则和使用和变量基本一样.自己回顾一下变量的命名规则.

函数体: 就是函数被执行之后要执行的代码

来我们来定义一个约x功能:

def yue():

    print("拿出手机")

    print("打开陌陌")

    print("找个漂亮的妹子")

    print("问她, 约不约啊?")

    print("oK. 走你!")

哦了定义完了. 但是这个时候去执行. 会发现什么都没有发生.  因为我只定义了一个函数. 但是还没有执行过这个函数.

函数的调用: 使用函数名可以调用函数, 写法: 函数名(), 这个时候函数的函数体会被执行

# 调用yue()函数

yue()

结果:

拿出手机

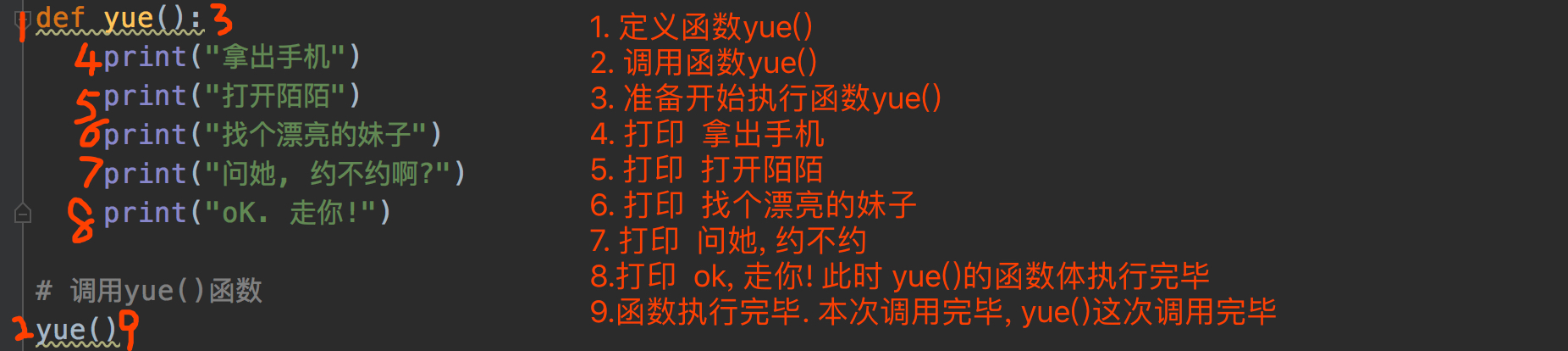
打开陌陌

找个漂亮的妹子

问她, 约不约啊?

oK. 走你!

    看一下执行过程:



    终于可以约了. 如果我还想约呢? 多次调用就可以了. 很方便.

# 调用yue()函数

yue()

yue()

yue()

yue()

yue()

    继续分析. 约完了之后你需要得到一个结果吧. 比如. 约完了得到了一个萝莉, 少妇, 大妈. 总得有个结果. 那么这个结果怎么来描述和获得呢? 这个就涉及到函数的返回值的问题.

**二. 函数的返回值**

        执行完函数之后. 我们可以使用return来返回结果.

函数中return的使用:

1. 函数中遇到return, 此函数结束, 不再继续执行.

def yue():

    print("约你")

    print("约我")

    print("约他")

    return

    print("约谁呀")    # 这句话不会被执行

yue()

2. 给函数的调用者一个访问结果

def yue():

    print("约你")

    print("约我")

    print("约他")

    return "美女一枚"

girl = yue()

print(girl)    # 美女一枚

3. 函数的返回值可以有多个结果

def yue():

    print("约你")

    print("约我")

    print("约他")

    return "美女一枚", "萝莉一枚"

girl = yue()

print(type(girl))   # tuple

总结一下:

1. 遇到return. 此函数结束, 函数后面的东西将不会再执行

2. return 返回值

        关于返回值:

            如果return什么都不写 或者 干脆不写return .那么返回的就是None

            如果return后面写了一个值. 则调用者可以接收一个结果

            如果return后面写了多个结果, 则调用者可以接收一个tuple, 调用者可以直接解构成多个变量

OK. 完美. 可以得到结果了. 但是我们的约的方式是不是有点儿问题呢?, 陌陌现在还能约到么? 约不到了吧. 该换探探了. 那过两天探探也死掉了呢? 是不是还会有一个替代品. 那你想. 有一个替代的. 你就需要去改一下代码. 是不是不太合适了. 最好的方式是不是想用什么约就用什么约? ok. 我们可以通过函数的参数来给函数传递一些信息.

**三. 函数的参数**

    参数, 函数在调用的时候指定具体的一个变量的值. 就是参数.  语法:

def 函数名(参数列表):

    函数体

    首先我们先把代码该一下. 能够实现上面的需求.

def yue(chat):

    print("拿出手机")

    print("打开"+chat)

    print("找个漂亮的妹子")

    print("约不约")

yue("陌陌")

yue("微信")

yue("探探")

结果:

拿出手机

打开陌陌

找个漂亮的妹子

约不约

拿出手机

打开微信

找个漂亮的妹子

约不约

拿出手机

打开探探

找个漂亮的妹子

约不约

    perfect. 搞定了. 我们在调用yue的时候给chat一个值. 然后再执行函数体.

关于参数:

1. 形参

写在函数声明的位置的变量叫形参.  形式上的一个完整. 表示这个函数需要xxx

2. 实参

在函数调用的时候给函数传递的值. 叫实参, 实际执行的时候给函数传递的信息. 表示给函数xxx

3. 传参

给函数传递信息的时候将实际参数交给形式参数的过程被称为传参.

def yue(chat):    # chat  形参

    print("拿出手机")

    print("打开"+chat)

    print("找个漂亮的妹子")

    print("约不约")

yue("陌陌")    # 实参

len("字符串")    # "字符串"在这里就是实参

print("麻花藤")    # "麻花藤"就是实参

**3.1 首先我们先看实参:**

    3.1.1 位置参数

    约到这里了. 有没有想过这个问题.  啥样的都约么?  哪里的都约么? 不一定吧. 比如, 我在北京, 我很寂寞, 我喜欢小姐姐. 强哥, 在哈尔滨, 很寂寞, 大妈就行了. 需求是不一样的. 而我们现在的函数没有这些功能.  那怎么办呢? 很简单, 多来几个参数就好了

def yue(chat, address, age):    # 形参

    print("拿出手机")

    print("打开"+chat)

    print("找个"+address+"附近漂亮的"+str(age)+"岁妹子")

    print("约不约")

yue("微信", "北京", 18)    # 实参

结果:

拿出手机

打开微信

找个北京附近漂亮的18岁妹子

约不约

    分析: 在访问yue()的时候, 我们按照位置的顺序分别把"微信", "北京", 18 赋值给 chat, address, age. 在传参过程中. 系统会默认按照位置把实参赋值到形参.

    练习: 编写函数, 给函数传递两个参数a, b. 比较a, b的大小, 返回 a, b中最大的那个数

    答案:

def my\_max(a, b):

    if a > b:

        return a

    else:

        return b

# 有点儿麻烦, 我们在这里学一个三元运算符.

def my\_max(a, b):

    c = a if a > b else b   # 当a>b成立返回a, 否则返回b

    return c

    3.1.2  关键字参数

    位置参数好不好呢?  如果是少量的参数还算OK, 没有问题. 但是如果函数在定义的时候参数非常多怎么办? 程序员必须记住, 我有哪些参数, 而且还有记住每个参数的位置, 否则函数就不能正常调用了. 那则么办呢? python提出了一种叫做关键字参数. 我们不需要记住每个参数的位置. 只要记住每个参数的名字就可以了

def yue(chat, address, age):

    print("拿出手机")

    print("打开"+chat)

    print("找个"+address+"附近漂亮的"+str(age)+"岁妹子")

    print("约不约")

yue(chat="微信", age=18, address="北京")       # 关键字参数.

结果:

拿出手机

打开微信

找个北京附近漂亮的18岁妹子

约不约

    搞定, 这样就不需要记住繁琐的参数位置了.

3.1.3 混合参数

可以把上面两种参数混合着使用. 也就是说在调用函数的时候即可以给出位置参数, 也可以指定关键字参数.

# 混合参数

yue("微信", age=18, address="上海")    # 正确.第一个位置赋值给chat, 后面的参数开始指定关键字.

yue(age="18", "微信", address="广州")    # 错误, 最开始使用了关键字参数, 那么后面的微信的位置就串了, 容易出现混乱

注意: 在使用混合参数的时候, 关键字参数必须在位置参数后面

综上: 在实参的角度来看. 分为三种:

1. 位置参数

2. 关键字参数

3. 混合参数,  位置参数必须在关键字参数前面

**3.2 在形参角度看**

3.2.1 位置参数. 按照位置来赋值, 到目前为止, 我们编写的函数都是这种

def yue(chat, address, age):

    print("拿出手机")

    print("打开"+chat)

    print("找个"+address+"附近漂亮的"+str(age)+"岁妹子")

    print("约不约")

3.2.2 默认值参数. 在函数声明的时候, 就可以给出函数参数的默认值. 在调用的时候可以给出具体的值, 也可以不给值, 使用默认值.

比如, 我们录入咱们班学生的基本信息. 通过调查发现. 我们班大部分学生都是男生. 这个时候就可以给出一个sex='男'的默认值.

def stu\_info(name, age, sex='男'):

    print("录入学生信息")

    print(name, age, sex)

    print("录入完毕")

stu\_info("张强强", 18)

注意, 必须先声明位置参数, 才能声明默认值参数.

3.2.3 函数参数--动态传参

动态接收位置参数, 在参数位置编写\*表示接收任意内容

def chi(\*food):

    print("我要吃", food)

chi("大米饭", "小米饭")

结果:

我要吃 ('大米饭', '小米饭')    # 多个参数传递进去. 收到的内容是元组tuple

动态接收参数的时候要注意: 动态参数必须在位置参数后面

def chi(\*food, a, b):

    print("我要吃", food, a, b)

chi("大米饭", "小米饭", "黄瓜", "茄子")

这时程序运行会报错. 因为前面传递进去的所有位置参数都被\*food接收了. a和b永远接收不到参数

Traceback (most recent call last):

  File "/Users/sylar/PycharmProjects/oldboy/fun.py", line 95, in <module>

    chi("大米饭", "小米饭", "黄瓜", "茄子")

TypeError: chi() missing 2 required keyword-only arguments: 'a' and 'b'

所以必须改写成以下代码:

def chi(\*food, a, b):

    print("我要吃", food, a, b)

chi("大米饭", "小米饭", a="黄瓜", b="茄子")   # 必须用关键字参数来指定

这个时候a和b就有值了, 但是这样写呢位置参数就不能用了. 所以. 我们要先写位置参数, 然后再用动态参数

def chi(a, b, \*food):

    print("我要吃", a, b, food)

chi("大米饭", "小米饭", "馒头", "面条")   # 前两个参数用位置参数来接收, 后面的参数用动态参数接收

那默认值参数呢?

def chi(a, b, c='馒头', \*food):

    print(a, b, c, food)

chi("香蕉", "菠萝")  # 香蕉 菠萝 馒头 (). 默认值生效

chi("香蕉", "菠萝", "葫芦娃")  # 香蕉 菠萝 葫芦娃 ()    默认值不生效

chi("香蕉", "菠萝", "葫芦娃", "口罩")    # 香蕉 菠萝 葫芦娃 ('口罩',) 默认值不生效

我们发现默认值参数写在动态参数前面. 默认值只有一种情况可能会生效.

def chi(a, b, \*food, c="娃哈哈"):

    print(a, b, food, c)

chi("香蕉", "菠萝")  # 香蕉 菠萝 () 娃哈哈   默认值生效

chi("香蕉", "菠萝", "葫芦娃")  # 香蕉 菠萝 ('葫芦娃',) 娃哈哈    默认值生效

chi("香蕉", "菠萝", "葫芦娃", "口罩")    # 香蕉 菠萝 ('葫芦娃', '口罩') 娃哈哈  默认值生效

这个时候我们发现所有的默认值都生效了. 这个时候如果不给出关键字传参. 那么你的默认值是永远都生效的.

顺序: 位置参数, 动态参数\*, 默认值参数

动态接收关键字参数

在python中可以动态的位置参数, 但是\*这种情况只能接收位置参数无法接收关键字参数. 在python中使用\*\*来接收动态关键字参数

def func(\*\*kwargs):

    print(kwargs)

func(a=1, b=2, c=3)

func(a=1, b=2)

结果:

{'a': 1, 'b': 2, 'c': 3}

{'a': 1, 'b': 2}

这个时候接收的是一个dict

顺序的问题, 在函数调用的时候, 如果先给出关键字参数, 则整个参数列表会报错.

def func(a, b, c, d):

    print(a, b, c, d)

# 关键字参数必须在位置参数后面, 否则参数会混乱

func(1, 2, c=3, 4)

所以关键字参数必须在位置参数后面. 由于实参是这个顺序. 所以形参接收的时候也是这个顺序. 也就是说位置参数必须在关键字参数前面. 动态接收关键字参数也要在后面

最终顺序(\*重点):

       位置参数 >  \*args > 默认值参数 > \*\*kwargs

       这四种参数可以任意的进行使用. 如果想接收所有的参数:

def func(\*args, \*\*kwargs):

    print(args, kwargs)

func("麻花藤","马晕",wtf="胡辣汤")

动态参数的另一种传参方式:

def fun(\*args):

    print(args)

lst = [1, 4, 7]

fun(lst[0], lst[1], lst[2])

fun(\*lst)   # 可以使用\*把一个列表按顺序打散

s = "臣妾做不到"

fun(\*s)     # 字符串也可以打散, (可迭代对象)

在实参位置上给一个序列,列表,可迭代对象前面加个\*表示把这个序列按顺序打散.

在形参的位置上的\* 表示把接收到的参数组合成一个元组

如果是一个字典, 那么也可以打散. 不过需要用两个\*

def fun(\*\*kwargs):

    print(kwargs)

dic = {'a':1, 'b':2}

fun(\*\*dic)

    函数的注释:

def chi(food, drink):

    """

    这里是函数的注释, 先写一下当前这个函数是干*什么的*, *比如我这个函数就是一个吃*

    :param food: 参数food是什么*意思*

    :param drink: 参数drink是*什么意*思

    :return: 返回的是什么东东

    """

    print(food, drink)

    return "very good"

**四. 函数的嵌套和作用域**

作用域: 一个变量能使用的作用范围

仔细分析以下代码.

a = 10 # 函数外面有个a

def func():

    a = 20 # 函数里面也有个a

    print(a) # 此时打印的结果?

func()  # 此时打印的结果?

注意, 在python中. 你可以认为变量的使用是遵循就近原则的.

在函数外面的变量被称为全局变量. 它的作用域是整个py文件. 在函数内部的变量被称为局部变量.作用范围仅限于函数内部.

我们可以通过globals()和locals()查看全局和局部作用域中的内容

a = 10

def func():

    a = 40

    b = 20

    def abc():

        print("哈哈")

    print(a, b)     # 这里使用的是局部作用域

    print(globals())    # 打印全局作用域中的内容

    print(locals())     # 打印局部作用域中的内容

func()

globals()查看全局作用域

locals()查看当前作用域

函数与函数之间可以互相的调用. 也可以互相的嵌套

def fun1():

    print(111)

def fun2():

    print(222)

    fun1()

fun2()

print(111)

# 函数的嵌套

def fun2():

    print(222)

    def fun3():

        print(666)

    print(444)

    fun3()

    print(888)

print(33)

fun2()

print(555)

**五. global和nonlocal关键字**

a = 100

def func():

    a += 1 # 报错.

    print(a)

func()

print(a)

注意报错那句话

这句话相当于a = a + 1 先计算右边. 右边会把全局变量a引入进来使用. 然后重新赋值给a. 但是. python中不允许函数内部改变外面变量的值. 这样做很不安全. python规定. 在函数内部想要修改全局变量. 必须使用global关键字把外面的变量引入才可以进行修改(赋值).

a = 100

def func():

global a

    a += 1 # 报错.

    print(a)

func()

print(a) # 101

相同的. nonlocal也是一样的操作. 它负责在内层函数中引入外层函数的局部变量

a = 10

def func1():

    a = 20

    def func2():

        nonlocal a # 有它没它两个结果

        a = 30

        print(a)

    func2()

    print(a)

func1()

结果:

加了nonlocal

30

30

不加nonlocal

30

20

**六. 内置函数(上)**

所谓内置函数. 其实我们已经学习过很多了. 本小结就给大家多聊聊关于内置函数的更多的知识点, 首先, 内置函数如果全部列出来大概60-70个. 有很多. 慢慢积累. 慢慢记就好

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| abs() | dict() | help() | min() | setattr() |
| all() | dir() | hex() | next() | slice() |
| any() | divmod() | id() | object() | sorted() |
| ascii() | enumerate() | input() | oct() | staticmethod() |
| bin() | eval() | int() | open() | str() |
| bool() | exec() | isinstance() | ord() | sum() |
| bytearray() | filter() | issubclass() | pow() | super() |
| bytes() | float() | iter() | print() | tuple() |
| callable() | format() | len() | property() | type() |
| chr() | frozenset() | list() | range() | vars() |
| classmethod() | getattr() | locals() | repr() | zip() |
| compile() | globals() | map() | reversed() | \_\_import\_\_() |
| complex() | hasattr() | max() | round() |  |
| delattr() | hash() | memoryview() | set() |  |

作用域相关:

    locals()      返回当前作用域中的名字

    globals()    返回全局作用域中的名字

迭代器相关:

    range()       生成数据

    next()         迭代器向下执行一次, 内部实际使用了\_\_next\_\_()方法返回迭代器的下一个项目

    iter()           获取迭代器, 内部实际使用的是\_\_iter\_\_()方法来获取迭代器

字符串类型代码的执行

    eval()        执行字符串类型的代码. 并返回最终结果

print(eval("2+2"))  # 4

n = 8

print(eval("2+n"))  # 10

def func():

    print(666)

eval("func()")  # 666

    exec()       执行字符串类型的代码

exec("""

for i in range(10):

    print(i)

""")

exec("""

def func():

    print("我是周杰伦")

func()

""")

  compile()   将字符串类型的代码变异. 代码对象能够通过exec语句来执行或者eval()进行求值

*'''*

*参数说明:*

 *1. re*s*ource 要执行的代码, 动态代码片段*

 *2. 文件*名*, 代码存放的文件名, 当传入了第一个参数的时候, 这个参数给空就可*以*了*

 *3. 模式*, *取值有3个,*

 *1*. *exec: 一般放一些流程语句的时候*

 *2*. *eval: resource只存放一个求值表达式.*

 *3*. *single: resource存放的代码有交互的时候. mode*应*为single*

*'''*

*cod*e*1 = "for* i *in range(10): print(i)"*

*c1* = *compile(*c*ode1, "", mode="exec")*

*exe*c*(c1)*

*cod*e*2 = "1+2+*3*"*

*c2* = *compile(*c*ode2, "", mode="eval")*

*a =* *eval(c2)*

*pri*n*t(a)*

*cod*e*3 = "name* *= input('请输入你的名字:')"*

*c3* = *compile(*c*ode3, "", mode="single")*

*exe*c*(c3)*

*pri*n*t(name)*

有返回值的字符串形式的代码用eval(). 没有返回值的字符串形式的代码用exec(). 一般很少用到compile()

输入和输出相关:

    input()    获取用户输入的内容

    print()    打印输出

内存相关:

    hash()    获取到对象的哈希值(int, str, bool, tuple)

    id()        获取到对象的内存地址

文件操作相关:

    open()    用于口打开一个文件, 创建一个文件句柄

模块相关:

    \_\_import\_\_()    用于动态加载类和函数

帮助:

    help()    函数用于查看函数或模块用途的详细说明

调用相关:

    callable()    用于检查一个对象是否是可调用的. 如果返回True, object有可能调用失败, 但如果返回False. 那调用绝对不会成功

查看内置属性:

    dir()    查看对象的内置属性, 方法. 访问的是对象中的\_\_dir\_\_()方法

基础数据类型相关:

    数字相关:

        bool()  将给定的数据转换成bool值. 如果不给值. 返回False

        int()     将给定的数据转换成int值. 如果不给值, 返回0

        float()  将给定的数据转换成float值. 也就是小数

        complex()  创建一个复数. 第一个参数为实部, 第二个参数为虚部. 或者第一个参数直接用字符串来描述复数

    进制转换:

        bin()    将给的参数转换成二进制

        otc()    将给的参数转换成八进制

        hex()    将给的参数转换成十六进制

    数学运算:

        abs()            返回绝对值

        divmode()     返回商和余数

        round()        四舍五入

        pow(a, b)    求a的b次幂, 如果有三个参数. 则求完次幂后对第三个数取余

        sum()    求和

        min()    求最小值

        max()    求最大值

和数据结构相关:

    列表和元组:

        list()        将一个可迭代对象转换成列表

        tuple()    将一个可迭代对象转换成元组

        reversed()    将一个序列翻转, 返回翻转序列的迭代器

        slice()    列表的切片

st = "大家好, 我是麻花藤"

s = slice(1, 5, 2)

print(st[s])

    字符串相关:

        str()            将数据转化成字符串

        format()     与具体数据相关, 用于计算各种小数, 精算等

# 字符串

print(format('test', '<20'))    # 左对齐

print(format('test', '>20'))    # 右对齐

print(format('test', '^20'))    # 居中

# 数值

print(format(3, 'b'))   # 二进制

print(format(97, 'c'))   # 转换成unicode字符

print(format(11, 'd'))   # 十进制

print(format(11, 'o'))   # 八进制

print(format(11, 'x'))   # 十六进制(小写字母)

print(format(11, 'X'))   # 十六进制(大写字母)

print(format(11, 'n'))   # 和d一样

print(format(11))   # 和d一样

# 浮点数

print(format(123456789, 'e'))   # 科学计数法. 默认保留6位小数

print(format(123456789, '0.2e'))   # 科学计数法. 保留2位小数(小写)

print(format(123456789, '0.2E'))   # 科学计数法. 保留2位小数(大写)

print(format(1.23456789, 'f'))   # 小数点计数法. 保留6位小数

print(format(1.23456789, '0.2f'))   # 小数点计数法. 保留2位小数

print(format(1.23456789, '0.10f'))   # 小数点计数法. 保留10位小数

print(format(1.23456789e+10000, 'F'))   # 小数点计数法.

        bytes()  把字符串转化成bytes类型

s = "你好"

bs = s.encode("UTF-8")

print(bs)

s1 = bs.decode("UTF-8")

print(s1)

bs = bytes(s, encoding="utf-8")     # 把字符串编码成UTF-8

print(bs)

        bytearray()    返回一个新字节数组. 这个数字里的元素是可变的, 并且每个元素的值得范围是[0,256), 这个没用

ret = bytearray('right',encoding='utf-8')

print(ret[0])

print(ret)

        memoryview()  查看bytes在内存中的情况

# 查看bytes字节在内存中的情况

s = memoryview("麻花藤".encode("utf-8"))

print(s)

        ord()    输入字符找带字符编码的位置

        chr()    输入位置数字找出对应的字符

        ascii()  是ascii码中的返回该值 不是就返回\u...

# 找到对应字符的编码位置

print(ord('a'))

print(ord('中'))

# 找到对应编码位置的字符

print(chr(97))

print(chr(20013))

# 在ascii中就返回这个值. 如果不在就返回\u...

print(ascii('a'))

print(ascii('好'))

        repr()    返回一个对象的官方表示形式

# repr 输出一个字符串的官方表示形式.

print(repr('大家好,\n \t我叫周杰伦'))

print('大家好我叫周杰伦')

# %r  %r用的就是repr

name = 'taibai'

print('我叫%r' % name)

    数据集合:

        dict()    创建一个字典

        set()     创建一个集合

        frozenset()    创建一个冻结的集合. 冻结的集合不能进行添加和删除操作

    其他相关:

        len()    返回一个对象中的元素的个数

        enumerate()    获取集合的枚举对象

lst = ["胡辣汤", "蛋花汤", "疙瘩汤"]

for index, el in enumerate(lst):

    print(str(index)+"==>"+el)

        all()      可迭代对象中全部是True, 结果才是True

        any()    可迭代对象中有一个是True, 结果就是True

print(all([1,2,True,0]))

print(any([1,'',0]))

        zip()    函数用于将可迭代的对象作为参数, 将对象中对应的元素打包成一个个元组, 然后返回由这些元组组成的开了表. 如果各个迭代器的元素个数不一致, 则返回列表长度与最短的对象相同.

l1 = [1,2,3,]

l2 = ['a','b','c',5]

l3 = ('\*','\*\*',(1,2,3))

for i in zip(l1,l2,l3):

    print(i)

sorted()    对可迭代对象进行排序操作(讲完lamda后再讲这个)

filter()     过滤(讲完lamda)

map()    会根据提供的函数对指定序列做映射(讲完lamda)

**重点总结:**

函数的基本语法:

def 函数名(形参1, 形参2.....):

函数体

return 值1, 值2, 值3...

函数名(实参)

形参: 函数声明的位置的变量. 准备用来接收数据的

实参: 在调用函数的时候给函数传递的具体的值.

返回值:

如果什么都不写. 默认在函数结束的时候返回None

如果只写了个return. 则遇见return结束函数运行. 返回None

如果写了return 值. 表示只有一个返回值.

如果写了return 值1, 值2, 值3.... 表示可以有多个返回值. 返回的时候会自动打包成元组

参数:

形参

1. 位置参数

2. 默认值参数

3. \*args: 动态接收位置参数

4. \*\*kwargs: 动态接收关键字参数

顺序: 位置 > \*args > 默认值 > \*\*kwargs

实参

1. 位置参数

2. 关键字参数

3. 混合参数

无敌传参:

def func(\*args, \*\*kwargs): # 什么都能传进去

pass

\*和\*\* 在形参表示聚合, 在实参表示打散

gobals() 查看全局作用域中的内容

locals() 查看当前作用域中的内容

global 将全局变量引入到局部来使用

nonlocal 将外层函数中的局部变量引入到内层函数中使用

当函数出现嵌套的时候注意看缩进.一层一层的慢慢看. 别慌

**练习题:**

写函数，检查传入字典的每一个value的长度,如果大于2，那么仅保留前两个长度的内容，并将新内容返回给调用者。

dic = {"k1": "v1v1", "k2": [11,22,33,44]}

PS:字典中的value只能是字符串或列表

写函数，此函数只接收一个参数且此参数必须是列表数据类型，

此函数完成的功能是返回给调用者一个字典，

此字典的键值对为此列表的索引及对应的元素。

例如

传入的列表为：[11,22,33]

返回的字典为 {0:11,1:22,2:33}

写函数，用户传入修改的文件名，与要修改的内容，执行函数，完成整个文件的批量修改操作。

读代码，回答：代码中,打印出来的值a,b,c分别是什么？为什么？

a=10

b=20

def test5(a,b):

print(a,b)

c = test5(b,a)

print(c)

读代码，回答：代码中,打印出来的值a,b,c分别是什么？为什么？

a=10

b=20

def test5(a,b):

a=3

b=5

print(a,b)

c = test5(b,a)

print(c)

写函数,传入函数中多个实参(均为可迭代对象如字符串,列表,元祖,集合等),

将每个实参的每个元素依次添加到函数的动态参数args里面.

例如 传入函数两个参数[1,2,3] (22,33)最终args为(1,2,3,22,33)

写函数,传入函数中多个实参(实参均为字典),将每个实参的键值对依次添加到函数的动态参数kwargs里面.

例如 传入函数两个参数{‘name’:’sylar’} {‘age’:1000}最终kwargs为{‘name’:’sylar’ ,‘age’:1000}

下面代码成立么?如果不成立为什么报错?怎么解决?

题目一：

a = 2

def wrapper():

print(a)

wrapper()

题目二：

a = 2

def wrapper():

a += 1

print(a)

wrapper()

题目三：

def wrapper():

a = 1

def inner():

print(a)

inner()

wrapper()

题目四：

def wrapper():

a = 1

def inner():

a += 1

print(a)

inner()

wrapper()

写函数，传入n个数，返回字典{‘max’:最大值,’min’:最小值}

例如：如:min\_max(2,5,7,8,4) 返回:{‘max’:8,’min’:2}

此题用到max(),min()内置函数

写函数，传入一个参数n，返回n的阶乘

例如:cal(7) 计算7\*6\*5\*4\*3\*2\*1

**作业题:**

HR人力资源管理.

1. 菜单: ("查看员工信息","添加员工信息", "修改员工信息", "删除员工信息", "退出")

2. 添加员工信息:

用户输入员工的基本信息(id, name, birthday, salary, input\_time),

将员工信息写入到文件emp.txt文件内

3. 修改员工信息:

显示所有员工信息. 然后让用户选择要修改的员工的id. 然后让用户输入员工的工资,

将员工的工资修改为用户输入的工资. 其余内容不做改动

4. 删除员工信息:

显示所有员工信息. 然后用户选择要删除的员工id, 根据用户输入的id删除该员工的全部信息

5. 查看员工信息:

显示出所有员工的基本信息.

按照这个顺序做可能会容易一点儿:

1, 2, 5, 4, 3