一 课程回顾

求两个浮点数的差 并返回他们的值

判断一个整数n能否同时被3 和 5 整除 ，如果能同时被3和5整除返回yes，否则返回No

定义一个函数，函数传一个列表[1,2,3，-1，-2,2],统计列表中负数的个数并返回结果

定义一个函数 ， 传入一个有重复元素列表 a=[1,2,3,4,4,5,6,2,1] ， 让函数返回的是一个无重复元素的列表

```
# def fn(a,b):
#     return a - b
# print(fn(1.2,1.1))

# def fn(n):
#     if n % 3 == 0 and n % 5 == 0:
#         return 'yes'
#     else:
#         return 'no'
# print(fn(10))

# def demo(a):
#     count = 0  # 存储变量
#     for i in a:
#         if i < 0:
#             count += 1
#     return count
#
# print(demo([1, 2, 3, -1, -2, 2]))


# def fn(a):
#     result = []
#     for i in a:
#         if i < 0:
#             result.append(i)
#     return result
# print(fn([1, 2, 3, -1, -2, 2]))


# def fn(a):
#     return list(set(a))
#
#
# print(fn([1, 2, 3, 4, 4, 5, 6, 2, 1]))
```

二 作用域和命名空间

2.1 作用域

域 ==》 区域

产生作用的区域

在程序中指的就是变量生效的区域

```
def fn():
    x = 1
print(x)
```

不同的变量 作用域也是不相同的

2.1.1 全局作用域

生命周期: 全局作用域在程序执行时创建，在程序执行结束时销毁

所有函数以外的都是全局作用域

在全局作用域中定义的变量都属于全局变量，全局变量可以在程序的任意位置被访问

```
y = 2
def fn():
    x = 1
    print(x)
    print(y)
fn()
print(y)
```

2.1.2 函数作用域

生命周期: 在函数调用时创建，在调用结束时销毁

每调用一次就会产生一个函数作用域

在函数作用域中定义的变量，都可以叫做局部变量，这个变量只能在函数的内部访问

```
def fn():
    a = 10
    print('a=',a)
# fn()
# fn()
# fn()

红线: 报错
黄线: 语法格式问题   可以运行，不会报错
```

```
def fn1():
    a = 30
    def fn2():
        print(a)
    fn2()
fn1()
```

```
def fn1():
    a = 20
    def fn2():
        a = 10
        print(a)
    fn2()
fn1()
```

使用变量的时候，会优先在当前作用域中寻找该变量，如果有就是用，没有就继续往上一级作用域寻找

```
def fn1():
    def fn2():  # fn2 函数内部的fn2  也就是函数作用域中
        pass
def fn2():  # fn2 全局中的fn2 也就是全局作用中
    pass
```

这两个fn2() 毫无关系

在函数中使用变量赋值时，默认都是为局部变量赋值

global关键字

```
a = 1
def fn():
    global a  # 声明在函数内部使用的这个a 是一个全局变量
    a = 2  #  变为全局变量
    print('内部a=',a)
fn()
print('外部a=',a)
```

```
总结:
    1.基本就是 里面能看到外面的， 但是外面不能看到里面的
    2.如果局部修改全局变量 可以加上一个global关键字
由内往外可以访问，但是由外往内事访问不了的
```

从里面可以看见外面，但是从外面看不见里面

这里的看见和看不见？==》 变量

变量究竟保存在哪里 ==》 命名空间

2.2 命名空间

指的是变量存储的位置

命名空间有几个?

全局作用域有全局的命名空间，用来保存全局变量

函数作用域有函数的命名空间，用来保存局部变量或者说是函数中的变量

实际上就是一个字典，是一个专门存储变量的字典

locals(): 获取当前作用域的命名空间

```
print(locals())

{'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__':
<_frozen_importlib_external.SourceFileLoader object at 0x0000021C786F6CD0>,
'__spec__': None, '__annotations__': {}, '__builtins__': <module 'builtins'
(built-in)>, '__file__': 'D:\\pycharm_文件\\pythonProject\\demo\\day_10\\作用域和命
名空间.py', '__cached__': None}
```

```
a = 1
def fn():
    pass

print(locals().keys())


dict_keys(['__name__', '__doc__', '__package__', '__loader__', '__spec__',
'__annotations__', '__builtins__', '__file__', '__cached__', 'a', 'fn'])
```

```
a = 1
def fn():
    b = 1
    print(locals())  # 打印当前的命名空间
fn()

{'b': 1}
```

```
a = 1
def fn():
    global b   # 将b 设为了全局变量  所以在函数内部 没有命名空间了
    b = 1
    print(locals())  # 打印当前的命名空间
fn()
```

```
locals()['b'] = 10  # 一般不适用
print(locals())
```

变量是创建在一个字典里面的

```
global ==》 全局

语法:

    def 函数名():
        global 变量名
        声明/操作变量名

还有一个叫做:
    nonlocal ==> 局部 内嵌函数中使用
    用来在函数或者其他作用域中使用外层(非全局)变量
    会从当前函数的外层函数开始一层层去查找名字，若是一直到最外层函数都找不到，就会抛出异常
    nonlocal声明的变量不是局部变量，也不是全局变量，而是外部嵌套函数内的变量
```

```
def f():
    a = 1
    def f2():
        nonlocal a
        a += 1  # a = a + 1 = 1 + 1 = 2
        print(a)
    f2()
f()

全局 = 大方的人 ==》 我的东西你们都可以用
局部 = 小气的人 ==》 我的东西只有我能用
```

2.3 高级用法

1.函数可以被引用(可以赋值)

```
def fn():
    print('fn')
f = fn
print(f,fn)
f()
```

2.函数可以作为参数传入另一个函数

```
def fn():
    print('fn')
f = fn


def fn2(x):
    print(x)  # fn 内存地址  print(fn)
    x()  # 调用x() 就相当于调用fn()  fn()
fn2(fn)
```

3.可以将函数作为返回值

```
def fn():
    print('fn')
f = fn


def fn2(x):   # x = fn
    return x  # reutrn fn
x = fn2(fn)  # fn2(fn) = fn    x = fn
x()
```

4.函数可以作为容器的元素

列表可以装任意数据的类型，包括函数

```
def fn():
    print('fn')
```

```
f = fn


def fn2(x):   # x = fn
    return x  # reutrn fn
x = fn2(fn)  # fn2(fn) = fn    x = fn
x()


li = [1,2,3,fn]
ff = li[3]  # 根据下标取函数fn，把fn 赋值给ff
ff()  # 其实就是fn()
```