

(一)函数的进阶

1.函数的多返回值

return x,y,z

2.函数的各种形参

1. 位置参数（必选参数）：
2. 调用时，传入的实参的个数和顺序必须和函数中定义的一样，否则会报错

```
def sample(a,b):  
    return a + b  
  
print(sample(1,2)) #3
```

缺少一个时：

```
def sample(a,b):  
    return a + b  
  
print(sample(1)) #TypeError: sample() missing 1 required positional argument: 'b'
```

多一个时：

```
def sample(a,b):  
    return a + b  
  
print(sample(1,2,3)) #TypeError: sample() takes 2 positional arguments but 3 were given
```

1. 默认参数（缺省参数）：

通过键值对定义，当该形参有实参传入时，则为传入的实参。没有时，则为默认的值，在传入参数时可以省略。

```
def sample(a,b,c=0):  
    return a + b + c  
  
print(sample(1,2))#3
```

当传入3时:

```
def sample(a,b,c=0):  
    return a + b + c  
  
print(sample(1,2,3))#6
```

3. 可变参数 (*args) (位置传递的不定长参数):

可以容纳任意个参数，且将这些参数根据传入的顺序依次组成一个元组传入函数中

```
def sample(*args):  
    a = args  
    print(type(args))#<class 'tuple'>  
    print(args)#(1, 2, 3)  
    print(a)#(1, 2, 3)  
  
sample(1,2,3)
```

4. 关键字参数:

传入的参数为"键 = 值".其中关键字参数的出入顺序可以改变，但必须在位置参数（可变参数）后。这样可以更直观，简洁，也清楚函数的作用。

```
def sample(a,b,c,d):  
    print(f'{a},{b},{c},{d}')
```

sample(1,2,d=3,c=4)# 1,2,4,3

5. 命名关键字参数(**kwargs):

传入任意数量的键值对参数，并且将这些参数组成一个字典返回到函数中。

```
def sample(**kwargs):  
    print(type(kwargs))#<class 'dict'>  
    print(kwargs)#['a' : 1, 'b' : 2, 'c' : 3]  
  
sample(a = 1,b =2,c=3)
```

1. 参数的顺序:

def sample(位置参数(必选参数),可变参数, 默认参数(缺省参数), 关键字参数, 命名关键字参数)
def sample(arg, a = 0, *args, kwarg, **kwargs)

```
def sample(a,b,c=0,*args,d,**kwargs):  
    print(f'a:{a},b:{b},c:{c},args:{args},d:{d},kwargs:{kwargs}')  
  
sample(1,2,3,5,d = 4,e = 6) #a:1,b:2,c:3,args:(5,),d:4,kwargs:{'e': 6}  
  
def sample(a,b,*args,c=0,d,**kwargs):  
    print(f'a:{a},b:{b},c:{c},args:{args},d:{d},kwargs:{kwargs}')  
  
sample(1,2,3,5,d = 4,e = 6)#a:1,b:2,c:0,args:(3, 5),d:4,kwargs:{'e': 6}
```

(二)匿名函数

1. 语法

lambda 参数,函数体（只能写一行代码）

lambda 关键字, 表示匿名函数

匿名函数只可以调用一次

参数为函数的形参, 即def sample(x,y,z)

函数体为函数的执行逻辑, 只能写一行, 不能写多行

一般使用在临时构建一个只使用一次的函数的场景

```
a = lambda x,y,z : x + y + z  
print(a(1,2,3))# 6  
  
a = lambda x,y,z : x + y + z  
print(a(1,z=3,y=6)) #10  
  
a = lambda x,y,z=0 : x + y + z  
print(a(1,y=6))#7  
print(z)#NameError: name 'z' is not defined
```

是因为这里是将函数中的计算逻辑传递给了a，而非整个函数

(三)文件的开关、读写

1.文件

数据存储在内存时，会随着电脑的关机而消失。要永久的保存数据，就要使用硬盘、u盘等容器进行存储，为了数据的管理和检索。引入了文件的概念。

2.编码

1. 定义

编码是一种规则集合，记录了内容和二进制之间的相互转换的逻辑

2. 编码技术

一种将内容和二进制相互转换的技术

3. 使用原因

因为计算机只能识别0和1，为了让计算机识别并保存内容。同时，将计算机中0和1反向编码成我们可识别的内容，所以引入了编码。

4. 常用编码：

utf-8

gbk

编码须统一，否则会乱码。

3.操作

1.打开文件

这里的f为文件对象

```
f = open(name,mode,encoding = 'utf-8'):
```

当 mode = 'r' 时，为只读文件。当name存在时，打开文件。当name不存在时，报错。

当 mode = 'w'时，为写入文件。当name存在时，打开文件，并且将文件的内容清空，从头开始写入。当文件不存在时，则创建一个文件，从头写入内容。

当 mode = 'a'时, 为追加内容到文件。当name存在时, 打开文件且将内容追加到内容的末尾。当name不存在时,创建一个新文件, 并且将内容追加到内容的末尾。

with open(name,mode,encoding = 'utf-8') as f:

with open()在代码执行结束时会将文件进行关闭, 而open()则不会, 需要配合close()来进行关闭

2.读取文件

read(num): 读取num个字节的内容, 如果没有指定, 则读取文件的全部内容。返回读取的内容。

text.txt文本内容为:

```
≡ text.txt
1  hello world
2  this is python
3  this is a text
```

```
f = open('text.txt','r')
fr = f.read()
print(fr)
f.close()
"""
    hello world
    this is python
    this is a text
"""
```

readlines():一次性读取所有行文本, 并返回一个列表, 每行之间用,隔开

```
with open("text.txt",'r') as f:
    fr = f.readlines()
    print(fr)
"""
['hello world\n', 'this is python\n', 'this is a text']
"""
```

readline():读取一行内容, 返回一行内容

```

with open("text.txt","r") as f:
    fr = f.readline()
    print(fr)

"""
    hello world

"""#其中有一行空格，是因为print()语句默认有换行

```

循环读取:

```

for line in open("text.txt","r"):
    line.strip(" ")
    print(line)
"""
for line in open("text.txt","r"):
    line.strip()
    print(line,end = '')
"""

```

3.写入文件

write():

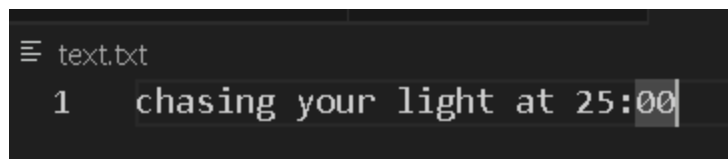
1.'w'模式下，如果文件存在，则清空文件内容，并从头向后写入内容；如果文件不存在，则创建文件并写入内容

```

f = open('text.txt','w')
f.write("chasing your light at 25:00")
f.flush()#将缓冲区的内容刷新到文件上
f.close

```

结果为:



```

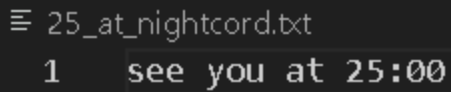
text.txt
1 chasing your light at 25:00

```

当文件不存在时:

```
f = open("25_at_nightcord.txt",'w')
f.write("see you at 25:00")
f.flush()
f.close()
```

结果为：

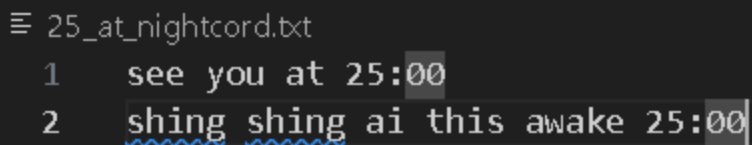


```
25_at_nightcord.txt
1  see you at 25:00
```

2.'a'模式下，如果文件存在，则将内容追加到文件内容的末尾；如果不存在，则创建文件，追加内容。

```
f = open('25_at_nightcord.txt','a')
f.write('shing shing ai this awake 25:00' )
f.flush()
f.close()
```

结果为：



```
25_at_nightcord.txt
1  see you at 25:00
2  shing shing ai this awake 25:00
```

如果不存在，则创建一个新的文件，将其写入。

```
f = open('notfound.txt','a')
f.write('notfound')
f.flush()
f.close()
```

3.关闭

f.close():关闭文件的占用，关闭时将缓冲区刷写到文件上

(四)异常处理：

1.异常

当检测到一个错误时，python编译器无法运行下去，并给出错误提示。这就是异常

2.使用原因

提前为可能出现异常的地方做好准备，并提醒异常的所在位置。当异常发生时，能够第一时间处理异常。

3.语法

1. 总体语法：

```
try:
    可能发生异常的代码
except:
    发生异常时执行的代码
else:
    没有发生异常时执行的代码
finally:
    无论是否发生异常都执行的代码
```

except后可以指定异常，如NameError,IndexError等异常，和父类（所有异常）Exception.或者将多个异常放入元组（NameError,IndexError）,当遇到指定的异常时立马执行except中的代码。

多个异常时：

```
try:
    f = open('abcd.txt','r')
    1/0
except (FileNotFoundError,ZeroDivisionError) as e:
    print(e)
    print("发现异常")
else:
    print("没有异常")
finally:
    print("程序执行完成")
"""
[Errno 2] No such file or directory: 'abcd.txt'
发现异常
程序执行完成
"""
```

单个异常时：


```

try:
    f = open('abcd.txt','r')
    1/0
except FileNotFoundError as e:
    print(e)
    print("发现异常")
else:
    print("没有异常")
finally:
    print("程序执行完成")
"""
[Errno 2] No such file or directory: 'abcd.txt'
发现异常
程序执行完成
"""

```

所有异常时:

```

try:
    1/0
    f = open('123.txt','r')
except Exception as e:
    print('发现异常')
    print(e)
else:
    print('没有异常')
finally:
    print('程序执行完成')
"""
发现异常
division by zero
程序执行完成
"""

```