

FSE598 前沿计算技术

模块 2 数据与数据处理 单元 3 编程技术 第 1 讲 函数与参数


讲座的英文版内容基于本书：

Y. Chen 《编程语言入门：C、C++、Scheme、Prolog、C# 和 Python 编程》(Introduction to Programming Languages: Programming in C, C++, Scheme, Prolog, C#, and Python)，第 6 版，Kendall Hunt Publishing Company，2019 年。

<https://www.public.asu.edu/~ychen10/book/IntroPl.html>

学习

□ 函数定义

- 类外的全局函数 
- 类中的成员函数（方法）

□ 不同的参数传递机制

□ 异常处理

□ 默认参数和关键词参数

函数与参数传递

函数/过程/子例程是必须显式调用的命名代码块。

使用函数的目的有两个：**抽象**和**重用**。

- **抽象**：构成一个概念单元的语句序列。
- **重用**：可以在程序中多个位置执行的语句。

函数与程序的其他部分通信可以通过

- 全局/静态变量
- 或 参数/返回值

形式参数和实际参数


- 在函数的定义(def) 中，给出的参数叫**形式参数**，它们是函数的局部变量。
- 调用函数时， 使用的参数叫**实际参数**。实际参数是调用者的变量/值。

函数示例

形式参数

```
def averageOfList(num):  
    sumOfNumbers = 0  
    for t in num:  
        sumOfNumbers = sumOfNumbers + t  
    avg = sumOfNumbers / len(num)  
    return avg  
def main():  
    a = averageOfList([19, 21, 46, 11, 18])  
    print("The average of List is ", a)  
    main()
```

实际参数

实际参数  形式参数

传递

- ❑ 这是一个计算平均值的函数
- ❑ num 是我们的参数。未定义类型。当列表被传递至 num 时，它就变成了一个列表类型
- ❑ 这个程序可以成功运行
- ❑ 但是.....
 - 如果我用非列表值调用，这个函数怎么样？！

Select C:\Program Files (x86)\Microsoft Visual Studio\Shared\

```
The average of List is 23.0  
Press any key to continue . . .
```



不同的语言实现参数传递的方式不同，包括以下传递机制，以及全局/静态变量：

- **值传递**：形式参数是函数中的局部变量。该参数通过初始化（复制）成为实际参数的值。它是实际参数的副本。
 - 优点：无副作用（安全、可靠）
 - 缺点：对于大型数据结构灵活性/功效性/效率欠佳
- **指针（引用）传递**：形式参数是实际参数的指针。
- **全局变量/静态变量**：没有显式的参数传递，但可以方便调用者和函数之间的通信，因为它们都可以访问全局或静态变量

- ❑ Python 通过与 Java 类似的方法传递参数
- ❑ 原始值类型和不可变类型按值传递
 - 又名 值传递
- ❑ 可变对象通过对象引用传递
 - 又名 指针传递（或引用传递）
 - 当我们通过对象引用传递时，这意味着我们可以变更对象，但不能重新对对象引用赋值。

参数与 Python

- ❑ 参数就像 Python 中的所有其他变量一样
- ❑ 我们可以在代码中添加语义，让编码器知道他们应该传递什么：
 - `def averageOfList(num:list):`
- ❑ 这不会作为语法强制执行
- ❑ 传递浮点数不会引发错误
- ❑ 这是由于 python 的信任哲学：
 - “在这里的都是成熟的成年人”
 - 基本上来看，Python 没有像类似于 Java 这样的偏执型语言的内置保护。
 - 这也意味着你应该学习并尊重 Python 的约定和文化

参数与 Python

- ❑ 但是.....如果你是一个不信任其他程序员的偏执狂.....
- ❑ Python 有适合你的命令
 - assert & isinstance

```
def averageOfList(num: list):  
    assert isinstance(num, list)  
    sumOfNumbers = 0  
    for t in num:  
        sumOfNumbers = sumOfNumbers + t  
    avg = sumOfNumbers / len(num)  
    return avg  
def main():  
    print("The average of List is",  
          averageOfList([19, 21, 46, 11, 18]))  
    main()
```

```
def isEven(value: int):  
    assert isinstance(value, int)  
    print(value)  
    if(value % 2 == 0):  
        return True  
    else:  
        return False
```


- ❑ assert 命令本质上是完整性检查
 - assert <condition>
 - assert <condition>,<error message>
- ❑ 从根本上来看，这是一种在不满足条件时强制代码出错的方法
 - 这是一条 ALL STOP 命令，并给出 AssertionError
- ❑ 通常用于测试和调试，而不是用于软件

- **异常：** 由已知的未知事件导致（强制）偏离程序的正常执行顺序。存在内部异常和外部异常。
- **内部异常：** 由来自 CPU **内部**向 CPU 发送的，目的在于引起注意的消息引起，例如被 0 除、溢出、数据不一致、非法指令。也称为软件相关异常。
- **外部异常：** 由来自 CPU **外部**向 CPU 发送的，目的在于引起注意的消息引起，例如内存不足、内存访问冲突、总线错误、设备忙。

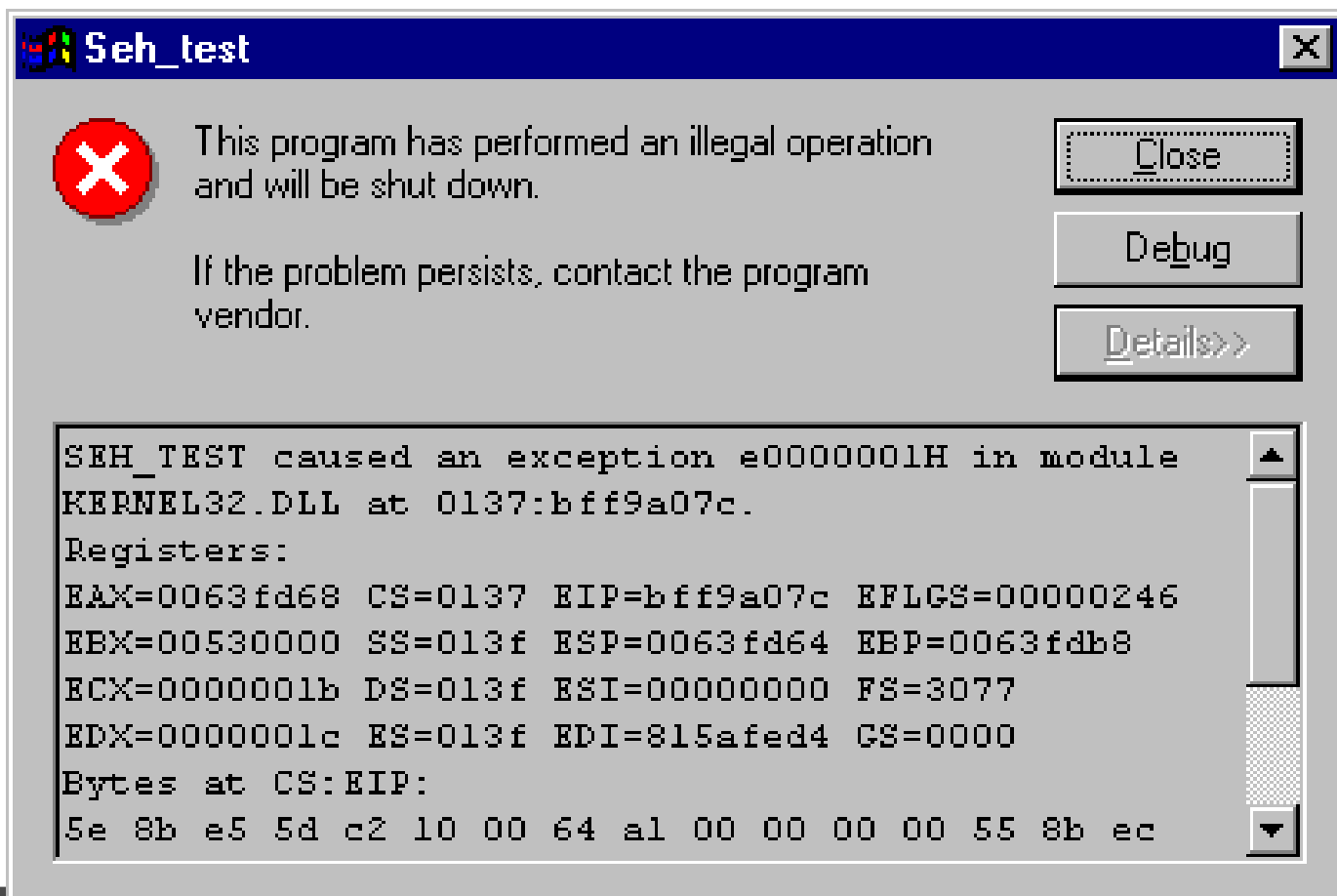
异常处理

将在不同层面对异常进行处理：

- 硬件层面：当异常发生时，硬件识别异常源并计算异常处理程序的入口地址。
- 操作系统层面：对于每个异常，编写一个简单的异常处理程，并将该处理程序存储在指定处理程序的入口地址。
- 编程语言层面：该语言提供异常，以便更详细地处理异常。
- 用户程序层面：程序员可以编写特定于应用程序的异常处理程序来处理异常。最好将执行所需功能的代码与处理异常的代码分开。
在这里我们将重点关注这一层面的异常处理。
- 每个上层都使用下层的处理程序，并且可以考虑额外的异常并添加处理程序。

未捕获异常


- ❑ 如果程序员编写的处理程序中没有覆盖某个异常，那么你可能会从较低层面的处理程序收到类似于这样的消息：



如果你正确编写异常处理程序，则可以避免这种情况。

异常处理——引发异常

- ❑ 后面我们会更详细探讨，但现在我们可以使用一些基本的异常处理来保护我们的代码
- ❑ 我们不使用 `assert`，而是 `raise` (引发)异常
 - 这在其他语言中称为 `throwing` (抛出) 异常
- ❑ 我们还可以使用 `isinstance` 来确认类型
- ❑ 但是现在我们 `raise` 引出一个异常，而不是使用 `assert` 作为一个全停命令



```
def isEven(value: int):  
    if not isinstance(value, int):  
        raise TypeError("value expects an integer")  
  
    print(value)  
    if (value % 2 == 0):  
        return True  
    else:  
        return False
```

异常处理——引发异常

- ❑ 有什么不同？
 - 程序员可以检查和处理异常
 - `assertion` 不能
- ❑ 这就是为什么 `assertion` 主要用于调试而不是软件

```
def isEven(value: int):  
    if not isinstance(value, int):  
        raise TypeError("value expects an integer")  
  
    print(value)  
    if (value % 2 == 0):  
        return True  
    else:  
        return False
```

异常

- ❑ Python 有大量内置的异常类，我们可以使用 `raise` 命令引发这些类
- ❑ 通常这些异常类足以清楚地表达
- ❑ 一些主要类：
 - 异常
 - 其他一切异常的基本通用基类
 - `TypeError`
 - `ArithmeticError`
 - `IndexError`
- ❑ <https://docs.python.org/3/library/exceptions.html>

❑ 函数的重载 (overloading)

- 是指基于不同的参数列表给相同的函数名多个定义。
- 不同的语言有不同的规则，但通常都需要参数列表变更
- C++、C# 和 Java 允许函数重载

❑ Python 中的函数不允许重载 (overloading)， 因为 Python 有动态类型系统，它允许将不同类型的数据传递给相同的参数

❑ 此外，Python 让我们能够设置默认参数

- 默认参数也可以称为可选参数

默认/可选参数

- 当我们定义参数列表中的某个参数时，我们可以使用 `=` 运算符为该参数设置一个默认值
- 如果带 `=` 的参数在函数调用中没有取值，则将其赋予默认值

```
def addNumbers(first, second, third = 0, fourth = 0):  
    return first + second + third + fourth  
  
addNumbers(1,2)  
addNumbers(1,2,3)  
addNumbers(1,2,3,4)
```


- 规则：
 - 默认参数按从左到右的顺序解析
 - 默认参数必须位于必要参数的右侧

有哪些规则？

- ❑ Python 有很多方法可以规避自己的规则
- ❑ Python 可以使用“关键词参数”
- ❑ 基本上是你确定你需要在函数调用中赋值的参数，并直接为其赋值的情况下
- ❑ 这也可以让你改变调用参数的顺序


```
# Function definition is here
def printinfo( name, age ):
    "This prints a passed info into this function"
    print "Name: ", name
    print "Age ", age
    return;
```

Normally, you call this way:

 printinfo("miki", 50)

```
# Function definition is here
def printinfo( name, age ):
    "This prints a passed info into this function"
    print "Name: ", name
    print "Age ", age
    return;
```

Now you can call printinfo function

 printinfo(age=50, name="miki")

使用带默认参数的关键词参数

- ❑ 在这里可以看到我有两个默认参数
- ❑ 但是我使用关键词参数来规避参数的正常分配顺序
- ❑ 我将第三个保留为 0，但给第四个赋值

```
def addNumbers(first, second, third = 0, fourth = 0):  
    print(first, second, third, fourth)  
    return first + second + third + fourth  
  
addNumbers(1, 2)  
addNumbers(1, 2, 3)  
addNumbers(1, 2, 3, 4)  
addNumbers( 1, 2, fourth=55)
```



1	2	0	0
1	2	3	0
1	2	3	4
1	2	0	55

可变参数函数：未知/任意/可变长度参数

- 有时候我们想要制作一个函数，但是不知道会有多少参数进入
- 例如
 - 我可能想将任意数量的数字加在一起
 - 我可能想要计算未知数量成绩的平均值
 - 我可能想对未知数量的学生进行处理操作
- 我们可以使用 * 作为参数名称的前缀来完成此操作
- 该参数现在将包含一个[含有这些值的元组](#)
 - 元组类似于列表，但它是一个不可变的集合
- 可变长度参数通常应该是参数列表中的最后一个
 - 除非你仅有用于默认和关键词的参数

addThemAll()

- ❑ 这是一个接受任意数量值的简单函数
- ❑ 要实现这一函数，我们可以使用 for 循环遍历参数中的列表
- ❑ 将它们全部加起来并返回总数

```
def addThemAll(*numbers):  
    total = 0  
    for value in numbers:  
        total += value  
    return total  
  
print(addThemAll(1,2,3))  
print(addThemAll(1,2,3,4,5,6,7,8,9))
```

6
45

重温 print()

- ❑ print() 实际上比我们之前所说的更广泛
- ❑ 它的完整语法如下所示：
 - `print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)`
- ❑ 可以通过关键词参数设置 sep、file 和 flush
 - sep
 - 可用于更改对象之间的分隔符，默认为 ' '
 - end
 - 可用于更改输出末尾显示的内容，默认为换行符 \n
 - file
 - 可用于更改输出的目标
 - 必须有一个 write(string) 方法
 - 可用于基本文件输出
 - flush
 - True 或 False，指定输出是刷新 (True) 还是缓冲 (False)