Note

Information

我负责的上机课资料会放在一个 Github 仓库: https://github.com/royess/intro-computation-materials

对内容的建议、勘误: 联系我或其他助教. 同时建议大家在 Github 上提 issue.

Python 解释器的工作方式

每次读入一行代码并处理. 默认一个语句不能换行.

一些特殊情况:

- 续行符 `\`, 说明下一行是本行的继续.
- 未配对的括号, 也认为下一行是本行的继续.
- 如所读行是某复杂结构 (例如 def, for, while...) 的头部, 会根据代码的退格形式继续读完整个结构, 之后再处理.

例子:

```
# 语句不能换行, 会报错
if a>0 and b>0 and c>0 and a+b>c
    and a+c>b and b+c>a:
    pass

# 括号不匹配自动延伸
if (a>0 and b>0 and c>0 and a+b>c
    and a+c>b and b+c>a):
    pass

# 用续行符号
if a>0 and b>0 and c>0 and a+b>c\
    and a+c>b and b+c>a:
    pass
```

程序的测试与调试

程序的测试与调试是开发中重要的一环. 通过测试与调试, 我们可以确认程序能正常工作, 满足需求.

测试 (Testing)

测试就是通过输入测试用例, 发现 bug 的存在.

测试的一些概念

程序往往由多个函数或多个模块组成. 我们把这些函数和模块称之为*单元*. 复杂的程序可以被拆分为大量的代码单元.

- 单元测试: 尽可能彻底地检查每个独立的代码单元, 确认其功能满足需求.
- 集成测试: 各代码单元测试完成后，逐步将其集成起来测试.

测试还可以分为黑箱和白箱两种方式:

- 黑箱测试: 不考虑程序内部结构, 只考虑功能. 测试时需要考虑基本情况, 边界情况和典型的错误情况等.
- 白箱测试: 根据内部结构进行测试, 保证所有执行路径都被覆盖到.

同学们设计简单的程序时, 可能不会用到这些方法, 但基本思想是有用的.

测试数据

对于简单的程序, 测试数据可以手动构造, 或者随机生成测试数据.

而对于复杂的程序, 会有更多测试技术. 可以参考甘老师的讲义.

调试 (Debugging)

调试就是找到 bug 的原因并修复.

简单的 debug 方式:

- 最最简单的方式: 插入 print 语句, 打印变量.
- 根据报错的位置和内容, 寻找错误.
- 在 Baidu, Google, StackOverflow, Github... 搜索遇到的问题和错误.
- 查看文档.

很多集成开发环境 (IDE) 提供了好用的 debugger. 这里介绍两种: Python 自带的 IDLE 和 PyCharm.

PyCharm 官方教程: https://www.jetbrains.com/help/pycharm/debugging-your-first-python-application.html#debug

断点调试

我们可以在程序的一些行上设置断点. 运行时遇到断点, 程序会自动暂停. 我们可以查看断点处变量的值, 来判断程序是否正常工作.

断点在运行过程中, 可以随时添加或者清除.

单步执行

除了设置断点, 我们还可以单行执行程序.

在 Python 程序中, 函数之间具有调用的层级结构 (堆栈).

在 `hello.py` 中, 函数调用层级: 主程序, `print_hello`, `random`...

根据函数调用关系, 单步执行分为以下几种:

- Step Over: 执行下一行, 不进入子函数.
- Step Into: 遇到子函数, 则进入子函数单步执行.
- Step Out: 跳出当前的函数, 返回调用函数的地方.

Hints for Exercises

Built-in Function `ord` and `chr`

问题: 输入一个未知的字母, 如何得到它后面的字母?

一种方法: 可以通过 Unicode code. 26 个字母的 Unicode code 是连续顺序排列的.

Python 中, 得到字符的 Unicode code 可以使用 `ord` 函数, 从 Unicode code 得到字符可以使用 `chr` 函数.

简单的例子:

```
>>> ord('A'), ord('B'), ord('Z')
(65, 66, 90)
>>> ord('a'), ord('b'), ord('z')
(97, 98, 122)
>>> chr(ord('A')+1)
'B'
```