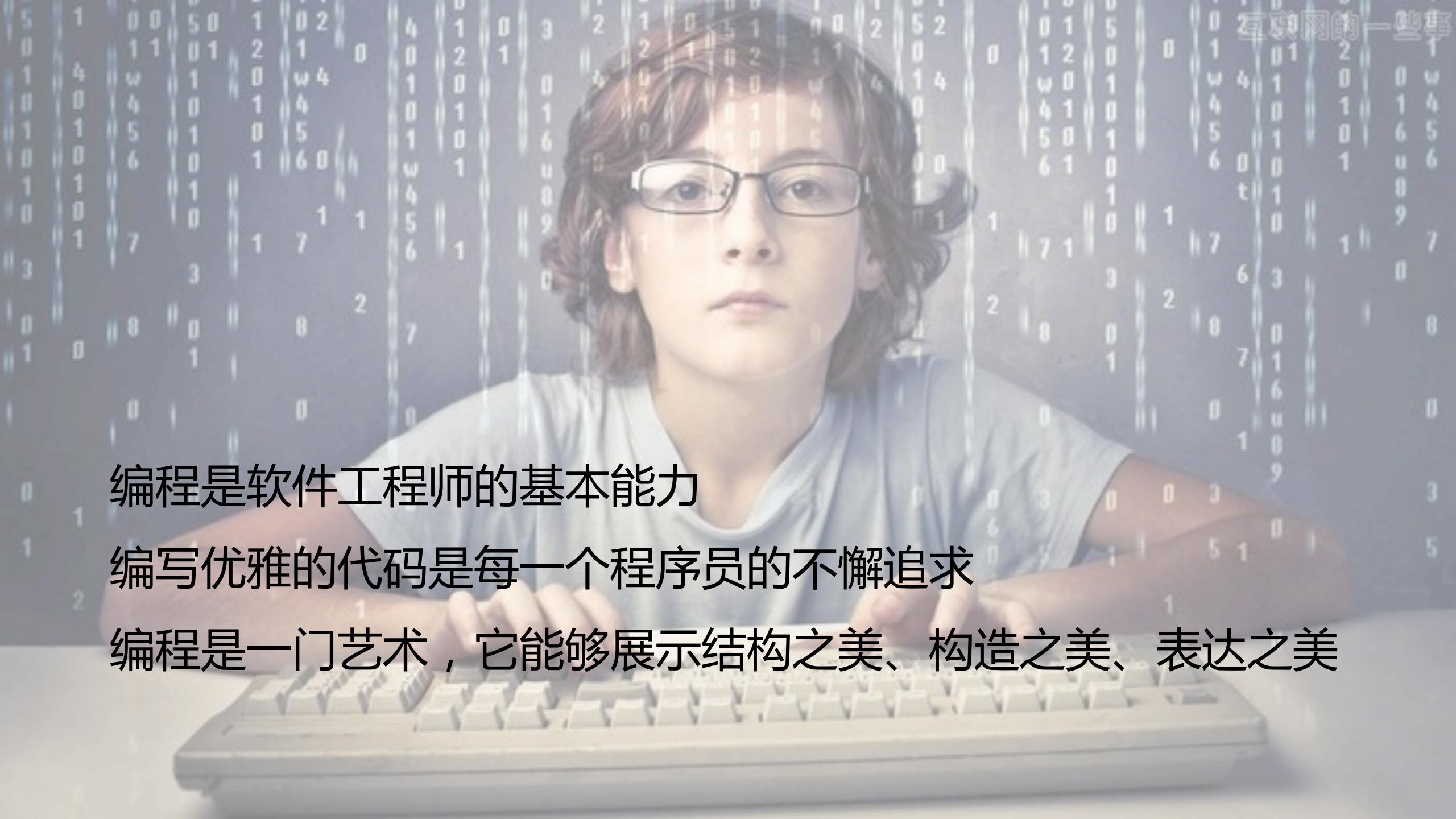




编程过程与规范

清华大学软件学院 刘强、陈华榕





编程是软件工程师的基本能力

编写优雅的代码是每一个程序员的不懈追求

编程是一门艺术，它能够展示结构之美、构造之美、表达之美

软件编程工作



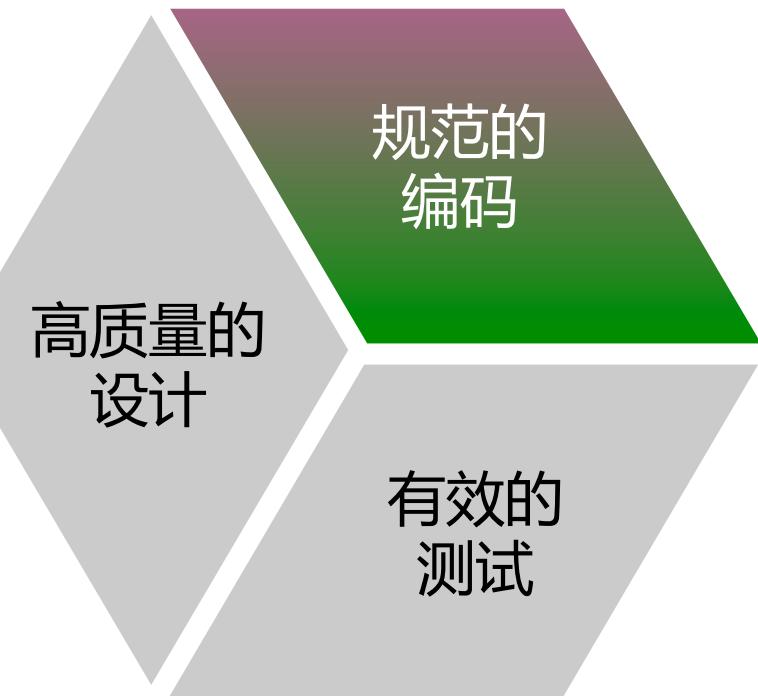
软件编程是一个复杂而迭代的过程，它不仅仅是编写代码，还应该包括代码审查、单元测试、代码优化、集成调试等一系列工作。



软件编程工作



高质量软件
开发之道



软件编程规范



软件编码规范是与特定语言相关的描写如何编写代码的规则集合。

现实

- 软件全生命周期的 70% 成本是维护
- 软件在其生命周期中很少由原编写人员进行维护

目的

- 提高编码质量，避免不必要的程序错误
- 增强程序代码的可读性、可重用性和可移植性



<https://github.com/google/styleguide>

Our [C++ Style Guide](#), [Objective-C Style Guide](#), [Java Style Guide](#), [Python Style Guide](#), [Shell Style Guide](#), [HTML/CSS Style Guide](#), [JavaScript Style Guide](#), [AngularJS Style Guide](#), [Common Lisp Style Guide](#), and [Vimscript Style Guide](#) are now available. We have also released [cpplint](#), a tool to assist with style guide compliance, and [google-c-style.el](#), an Emacs settings file for Google style.

If your project requires that you create a new XML document format, our [XML Document Format Style Guide](#) may be helpful. In addition to actual style rules, it also contains advice on designing your own vs. adapting an existing format, on XML instance document formatting, and on elements vs. attributes.

Python编程规范：程序模板

```
#!/usr/bin/env python3  
# -*- coding: utf-8 -*-  
  
#  
__author__ = 'YourName'
```

被直接执行的文件建议增加此行，被导入的模块请忽略

文件的基本头部

```
import sys
```

模块的导入总应该放在文件顶部

```
def main(argv):
```

主功能建议创建main函数来执行

```
    """Do something..."""
```

函数、类等应遵循一定的注释规范

```
    return 0
```

```
if __name__ == '__main__':  
    exit(main(sys.argv[1:]))
```

即使打算被直接执行的文件也应是可导入的，
这样当模块被导入时主程序就不会被执行

Python编程规范：注释

- 形式1：由 `#` 开头的“真正的”注释，说明选择当前实现的原因以及这种实现的原理和难点；



```
def gold_divide(n):
    # 黄金分割点比例为( $\sqrt{5}-1$ )/2≈0.618034

    return n * 0.618 # 直接取0.618以加速
```

- 形式2：文档字符串，说明如何使用包、模块、类、函数（方法），甚至包括使用示例和单元测试。



```
def gold_divide(n):
    """ Get gold divide value of n.

    Args:
        n: input number

    Returns:
        A float value
    """

    return n * 0.618
```

```

def main(argv):
    """
    函数main, 返回0表示操作成功, 返回-1表示失败
    """

try:
    # 以二进制只读方式打开文件
    with open('cat.pic', 'rb') as fin: # fin是输入文件
        # 将文件指针移动到文件尾
        if fin.seek(0, 2) != 400 * 400 * 3: # 判断文件尾位置是否等于400*400*3
            print("输入文件 cat.pic 不符合格式要求") # 如果不等, 显示错误信息
            return -1
        fin.seek(0) # 将文件指针移动到文件开头
    try:
        # 以二进制写方式打开文件
        with open('cat2.pic', 'wb') as fout: # fout是输出文件
            for i in range(400): # 从图像的第1行到第400行循环
                for j in range(400): # 从每行的第1列到第400列循环
                    # 从输入文件中读入每像素的RGB值
                    b = ord(fin.read(1))
                    g = ord(fin.read(1))
                    r = ord(fin.read(1))
                    # 按照公式 Y=0.299R+0.587G+0.114B 计算灰度值
                    y = (299 * r + 587 * g + 114 * b) / 1000
                    fout.write(chr(y)) # 将计算出来的灰度值写到输出文件中去
    except IOError as e:
        print("打开文件 cat2.pic 时错误") # 如果打开失败则显示错误信息
    return -1
except IOError as e:
    print("打开文件 cat.pic 时错误") # 如果打开失败则显示错误信息
return -1
return 0 # 返回0表示正确处理完毕

```

注释仅仅是语句的重复解释，没有任何价值



你明白这段代码的作用吗？

Python编程规范：注释



学会只编写够用的注释，过犹不及，重视质量而不是数量。

- 好的注释解释为什么，而不是怎么样
- 不要在注释中重复描述代码
- 当自己在编写密密麻麻的注释来解释代码时，需要停下来看是否存在更大的问题
- 想一想在注释中写什么，不要不动脑筋就输入
- 写完注释之后要在代码的上下文中回顾一下，它们是否包含正确的信息？
- 当修改代码时，维护代码周围的所有注释

```
def main(argv):
    """
    主函数，返回0表示成功
    """
try:
    with open('cat.pic', 'rb') as fin:
        if fin.seek(0, 2) != 400 * 400 * 3: # 判断文件长度是否符合格式要求
            print("输入文件 cat.pic 不符合格式要求")
            return -1
        fin.seek(0)
        try:
            with open('cat2.pic', 'wb') as fout:
                # 下面是图像转换的算法实现。彩色图像到灰度图像的转换主要利用
                # RGB色彩空间到YUV色彩空间的变换公式来取得灰度值Y，公式是
                #  $Y = 0.299R + 0.587G + 0.114B$ 
                for i in range(400):
                    for j in range(400):
                        b = ord(fin.read(1))
                        g = ord(fin.read(1))
                        r = ord(fin.read(1))
                        y = (299 * r + 587 * g + 114 * b) / 1000
                        fout.write(chr(y))
                except IOError as e:
                    print("打开文件 cat2.pic 时错误")
                    return -1
            except IOError as e:
                print("打开文件 cat.pic 时错误")
                return -1
    return 0
```



Python编程规范：注释



```
# -*- encoding: utf-8 -*-

"""
斐波那契数列
"""

def fibonacci(position):
    """获得斐波那契数列指定位置上的数。
    斐波那契数列是形如1, 1, 2, 3, 5, 8, 11, ... 的数列，通项公式为
    F_0=F_1=1, F_n=F_{n-1}+F_{n-2}, n>=2。
    Args:
        position: 需要获得的指定位置
    Returns:
        一个数，即F_{position}
    """
    if position < 2:
        return 1

    previousButOne = 1
    previous = 1
    result = 2
    for n in range(2, position):
        previousButOne = previous
        previous = result
        result = previous + previousButOne
    return result
```

The image shows a comparison between Python source code and its generated documentation. On the left is the original Python code for a `fibonacci` function. On the right is the generated documentation in the form of a pydoc-style page. The page has a blue header with the title `fibonacci`, a white main content area with the docstring, and an orange sidebar. The sidebar contains the function's name, its description, its arguments, and its return value. An orange arrow points from the code to the generated documentation.

fibonacci

斐波那契数列

Functions

`fibonacci(position)`
获得斐波那契数列指定位置上的数。
斐波那契数列是形如1, 1, 2, 3, 5, 8, 11, ... 的数列，通项公式为 $F_0=F_1=1, F_n=F_{n-1}+F_{n-2}, n\geq 2$ 。

Args:
`position`: 需要获得的指定位置

Returns:
一个数，即 $F_{position}$

Python编程规范：命名



好的名字一目了然，不需要读者去猜，甚至不需要注释。



- Python库的命名约定有点混乱，因此很难使之变得完全一致，不过还是有公认的命名规范。
- 新的模块和包（包括第三方的框架）必须符合这些标准，但对已有的库存在不同风格的，保持内部的一致性是首选的。

Python编程规范：命名

不要编写需要外部文档支持的代码，这样的代码是脆弱的，要确保你的代码本身读起来就很清晰。

编写自文档化的代码

- 唯一能完整并正确地描述代码的文档是代码本身
- 编写可以阅读的代码，其本身简单易懂



Python编程规范：命名

```
def fval(i):
    ret = 2
    n1 = 1
    n2 = 1
    i2 = i - 3
    while i2 >= 0:
        n1 = n2
        n2 = ret
        ret = n2 + n1
        i2 -= 1
    return 1 if i < 2 else ret
```



Python编程规范：命名

```
def fval(i):
    ret = 2
    n1 = 1
    n2 = 1
    i2 = i - 3
    while i2 >= 0:
        n1 = n2
        n2 = ret
        ret = n2 + n1
        i2 -= 1
    return 1 if i < 2 else ret
```



```
def fibonacci(position):
    if position < 2:
        return 1

    previous_but_one = 1
    previous = 1
    result = 2
    for n in range(2, position):
        previous_but_one = previous
        previous = result
        result = previous + previous_but_one
    return result
```

Python编程规范：语句



`print("Hello, world!");` 不要在行尾加分号

```
for i in range(n):  
    result += i
```

不要使用制表符 (tab) 进行缩进，千万不要混用tab与空格，应使用4个空格进行缩进

```
class book shelf:  
    pass
```

类名应为驼峰风格且首字母大写，这里应该是BookShelf

bookShelf = BookShelf()

变量名应为下划线风格，这里应该是bool_shelf

f = open('output.txt', 'w')

使用文件时应注意显式地调用close()，或使用with：

```
if n > 5: print(n)  
else: n = 6
```

每行只写一条语句

```
with open('output.txt', 'w') as f:  
    pass
```

Python编程规范：语句



import 语句应遵循的原则：

- import 次序：先 import Python 内置模块，再 import 第三方模块，最后 import 自己开发的项目中的其它模块；这几种模块中用空行分隔开来。
- 一条 import 语句 import 一个模块。
- 当从模块中 import 多个对象且超过一行时，使用如下断行法（py2.5以上版本）：
`from module import (obj1, obj2, obj3, obj4, obj5, obj6)`
- 不要使用 from module import *，除非是 import 常量定义模块或其它你确保不会出现命名空间冲突的模块。

Python编程规范：语句



Python还有很多灵活的用法，使用时应注意删繁就简。

- Python的列表推导简洁方便，但仅应在简单的情况下使用

✓ `[s['name'] for s in students if s['age'] > 18]`

✗ `[(x, y) for x in range(m) for y in range(n) if x ** 2 + y ** 2 >= 4]`

- Python的条件表达式是if语句的简短语法规则，但同样仅应在简单的情况下使用

✓ `return s['name'] if s['age'] > 18 else s['nickname']`

✗ `return s['name'] if s['age'] > 18 else s['nickname'] if s['age'] > 14 else 'anonymous'`

- Python还有一些很“酷”的特性，但奇技淫巧的代码会难以开发、调试、维护

Python编程规范 : 更多完整内容



请参考Google Python Style Guide :

<http://google.github.io/styleguide/pyguide.html>

Google Python Style Guide

以及Python之禅道 (The Zen of Python) :

打开Python交互式解释器并输入import this即可查看。

`import this`



谢谢大家！

THANKS

