总体流程大概分为以下几步：

1. 数据导出（file\_deal.py）：将下载好的包里对应格式的文件抽出（比如cpython就只抽.py文件），读取每个文件的文本内容，形成初始数据集。
   1. 函数1：get\_file——遍历原始文件夹，把所需的文件类型抽出来，汇总在同一文件夹下以便后续处理
   2. 函数2：file\_to\_df——依次打开上一步汇总的文件夹下的每一个文件，记录文件内容及标签，存为dataframe
2. 数据清洗：
   1. 做直方图观察每个类别下的数据分布，保证整个训练集是相对无偏的
   2. 极端样本剔除：去掉text长度100以下和10万以上的样本
   3. 无偏处理：.js文件的数量和分布明显与其他三种差异很大，因此在所有.js文件中，按照原始分布随机抽取500个样本
   4. 测试集和训练集生成：以每个类别8:2的比例切分测试和训练集，同时，另外加入100个.js样本，保证测试集和训练集的分布不雷同，以更好的检验模型稳定性。
   5. 生成结果：train\_set.csv test\_set.csv
3. 分词（word\_cut.py 函数word\_cut）：
   1. 首先用strip和replace去掉换行符、制表符等
   2. 直接使用jieba分词工具，对于编程语言的识别，标点符号反而很重要，比如各语言的注释符号不同，所以分词要小心，不能随便去停用词和标点符号。
   3. 分词过程中过滤空格等。
   4. 考虑到不同人写注释的习惯，可能会出现连续几十个注释符号之类的情况，把分词结果中长度大于25的直接过滤掉。
   5. 生成结果：train\_set\_cuted.csv test\_set\_cuted.csv
4. 词频统计（x\_generate.py 函数one\_hot\_x）：
   1. 训练集：对训练集做词频统计，过滤掉总词频为1的低频词，生成最后的词库（共31783个词）
   2. 测试集：以训练集生成的词库为标准，统计测试集中，词库词语出现的频数
   3. 生成结果：train\_x\_count.csv test\_x\_count.csv
5. 模型评价标准的定义（classifier.py 函数performance）：三个标准
   1. 总体精确度：即所有预测样本中，预测正确的样本所占比例
   2. 分类别查准率
   3. 分类别召回率
6. 模型1：one-hot + 朴素贝叶斯
   1. One-hot（独热矩阵）：以上述词库为基准，对于词库中的每一个词，如果某个样本中，出现了该词，则记为1，如果没有出现，则记为0。以该方式遍历所有样本，即可生成一个0-1矩阵，作为输入模型的features.
   2. 生成的X：train\_x\_one\_hot，test\_x\_one\_hot(这里没有导出csv)
   3. 朴素贝叶斯分类（classifier.py 函数naive\_bayes）：因为输入值均为0和1，所以显然最适合的是二项分布，即BernoulliNB.
   4. 训练和预测：使用训练集训练模型，然后分别对训练集和测试集做预测，用performance函数分析样本内和样本外表现。
   5. 生成结果：perfor\_train\_nb， perfor\_test\_nb， txt输出nb\_out.txt（在out文件夹下）
   6. 准确度：
      1. 样本内：总体准确率 0.89；查准js较低，召回c和py较低
      2. 样本外：总体准确率0.76；查准js较低，召回c极低，py较低
7. 模型二：词频/文本长度 + PCA + 决策树
   1. 词频/文本长度：独热方式忽略了词频的重要性，一些反复出现的关键词对我们识别特定的编程语言是有很大帮助的。因此，这一方法旨在保留词频给出的信息。此外，由于不同文本的长度不同，直接比较词频是不合适的，所以需要用词频/文本长度的方式进行统一。
   2. 词频/文本长度输出：train\_x\_count\_percent，test\_x\_count\_percent
   3. PCA（x\_generate.py 函数pca）：独热方式下，由于都是0-1矩阵，不适合做降维处理，所以只能选择输入维度达到3万多维的features进行训练和预测。但文本/词频长度方式下，是可以进行主成分分析降维的。因此，这里对词频/文本长度的输出结果做PCA进行降维。
   4. PCA参数优化：既然要做PCA，就涉及到保留多少个PC作为输入值的问题。以最终保留的PC对总体方差的衡量百分比为输入参数，最终决策树分类模型给出的样本外精确度为优化目标，分别观察keep\_vars = [0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 0.98]下的表现。最终选定0.8为最合适的参数
   5. 决策树分类及预测（classifier.py 函数decision\_tree）：
      1. 样本内：准确率100%（在维度比较低的时候，决策树出现样本内100%非常正常），查准召回皆为100%
      2. 样本外：准确率0.88， 查准各类别基本均衡，召回java较低。
   6. 结果输出：out文件夹 dt\_out.txt
8. 模型三：Word2vec + 决策树
   1. Word2vec: 使用google开发的word2vec模型，将训练数据中的分词结果直接转化为词向量模型（x\_generate.py 函数word2vec），然后，遍历训练样本，对每一个样本中的每一个词，检索模型得到其词向量，然后加和，除以整个文档的长度，即可得到代表了整个文档特征的输入向量（word\_cut.py 函数 add\_word\_to\_sentence）。
   2. 将上面得到的输入向量作为features来训练决策树模型，然后进行预测
   3. 模型表现：
      1. 样本内：准确率0.91，查准js较低，召回c和py较低
      2. 样本外：准确率0.82， 查准js和py均较低，召回c和py较低
   4. 结果输出：out文件夹 wvdt\_out.txt
9. 关于下一步的改进：
   1. 参数优化：时间关系，有一些模型参数没来得及优化，比如说决策树的树深度，word2vec的忽略词频数，最终生成的向量维度等，这些优化工作皆可作为下一步工作。
   2. 分词过程的细化：大部分代码文件中，都会有注释存在，而这些注释与代码本身的语法结构是完全不相干的。所以如果能在分词过程中将注释内容直接过滤掉，应该可以进一步地提升模型表现。可以考虑针对每一种文件类型，去逐行识别其注释符号，比如python的 #，‘’’, ””” 等，然后保留注释符号，去掉注释内容。
   3. 样本搜集：目前使用的训练样本其实是有偏的，js样本分布明显和其他三种不一样。如果时间充裕，应该做到平衡训练样本的分布，这一点对于朴素贝叶斯模型来讲尤其关键，因为根据贝叶斯法则，每一类样本出现的概率会直接影响模型的预测结果，如果在这方面的假设不合理，那模型必然会有偏。
   4. 分类模型：研究主要用到的是朴素贝叶斯和决策树，其中决策树的表现比较好，进一步可以考虑使用随机森林、xgboost等来改善其表现。