Workflow and debugging

10-601 Recitation
Fall 2020
How to write code, in a better way?

Why don’t we start with a good IDE or text editor?

- IDEs: Spyder, PyCharm, Eclipse, CLion, XCode, ...
- Text editors: Visual Studio Code, Sublime Text, Atom, ...

Benefits:

- Auto-completion, Syntax checks, Formatting options
- Interactive environment, Debugger, Command line support
An example workflow

*Modularity*: think about the conceptual flow rather than the concrete implementation.

Don’t do this!
An example workflow

Modularity: think about the conceptual flow rather than the concrete implementation.

Read data

Train a model

Predict on train data

Compute train error
An example workflow

*Modularity*: think about the conceptual flow rather than the concrete implementation.

```python
def read_data(input_file):
    # Read data

input_file = sys.argv[0]
x_train, y_train = read_data(input_file)

def train(x_train, y_train):
    # Train a model

model = train(x_train, y_train)

def predict(model, x_train):
    # Predict on train data

y_pred = predict(model, x_train)

def error(y_train, y_pred):
    # Compute train error

train_error = error(y_train, y_pred)
```
An example workflow

*Modularity*: think about the conceptual flow rather than the concrete implementation.

- Separation of complex functionalities
- Better readability
- Easier debugging and maintenance
- Facilitate code reuse
An example workflow

Write your awesome code, *as well as your awesome comments.*

```python
def read_data(input_file):
    """Code author: Matt Gormley"""
    with open(input_file, 'r') as f:
        lines = f.readlines()

    # Drop header
    lines = lines[1:]
    # Convert to list of lists of strings
    rows = [l.strip().split('	') for l in lines]
    # Convert to numpy array
    return np.array(rows)
```
An example workflow

After finishing your code, *do some unit tests right away, if possible.*

- Unit test (noun.): test for the individual components of a software [Modularity!]
- A good practice is to separate tests from the main program

```python
def transpose(A):
    # do something plausible
    return A

A = np.array([[1, 2], [3, 4]])
A_true_transpose = np.array([[1, 3], [2, 4]])
assert np.array_equal(transpose(A), A_true_transpose)
```
An example workflow

After finishing your code, an error occurs!

- Understand the error message
  - Google it if you don’t understand the error
  - Quickly locate your error from the error message
An example workflow

After finishing your code, an error occurs!

- Understand the error message
  - Google it if you don’t understand the error
  - Quickly locate your error from the error message

- Use a debugger
  - PDB for Python, JDB for Java, GDB/LLDB for C++; Debugging tools in IDE
  - Print relevant variables to see if the program works as expected
  - Logging in Python/Java: advanced print for debugging
Coding Style Conventions

- Why have good style?
- Not required for the class but ....
  - Make it easier when you are coding through ideas
  - Makes debugging easier
- What’s a good coding style to follow?
  - Python PEP 8 Style Guide
  - Full guide can be found if you just google that
Spacing and Maximum Line Length

- Tabs vs Spaces
  - 1 tab or 4 spaces
  - Stay consistent, don’t mix!!!

- Try using blank lines to separate different chunks of code
- Do not have more than 79 characters per line
  - Usually text editors will tell you how many characters there are on a certain line

```python
# Hanging indents should add a level.
foo = long_function_name(
    var_one, var_two,
    var_three, var_four)
```
Importing and Comments

- Import packages on separate lines, not on same line!
- Do not import wildcard
  - Never do: “from os import *”
- Comments are super important and useful!
  - When looking back it's easier to figure out what certain chunks of code do
  - Helps with debugging
  - Use # for commenting out a single line

# Correct:
import os
import sys

# Wrong:
import sys, os
Variable Naming Conventions

- Do not give variables and functions non descriptive names:
  - Bad examples: “x”, “a”, “myFun”
  - Having descriptive variable and function names is similar to good commenting
  - Easier coding and debugging
  - You won’t forget what a variable is being used for if it’s descriptive

- Constant names should be in all caps
- In general DON’T USE global variables
Debugging: Common Python Errors

- SyntaxError: invalid syntax
  - Forgetting the parens around the arguments to print
  - Forgetting the colon at the end of the condition in an if statement
  - Trying to use a reserved word as a variable name
- IndentationError: expected an indented block
  - Forgetting to indent the statements within a compound statement (such as the bodies of if and for)
  - Forgetting to indent the statements of a user-defined function
- IndexError: list index out of range
  - Trying to access an item in a list at an invalid index
Debugging: Common Python Errors

- KeyError
  - Trying to access a non-existing key in a dict
- TypeError: 'list' object cannot be interpreted as an integer
  - Forgetting the len() call in a for loop statement.
- IndexError
  - Trying to access an item in a list at an invalid index
- UnboundLocalError: local variable 'foobar' referenced before assignment
  - Using a local variable (with the same name as a global variable) in a function before assigning the local variable