Managing Large Code Projects

15-110 – Wednesday 11/11

Learning Goals

- Implement **helper functions** in code to break up large problems into solvable subtasks
- Recognize the four core rules of **code maintenance**
- Use the **input** command and **try/except** structures to handle direct user input in code
- Learn how to install and use **external modules**

Unit Overview

New Unit: CS as a Tool

Our next unit focuses on how computer science can be used to benefit other domains.

We'll investigate three different applications of computer science: **data analysis**, **simulation**, and **machine learning**.

These three applications share a core idea in common: all three **organize data** to **help people answer questions**.

Schedule for Unit 5

The schedule for this unit will be **staggered**.

The first week (Fri 11/13 – Wed 11/18) will focus on how the applications **organize data**.

The second week (Fri 11/20 – Mon 11/30) will focus on how the applications **find answers**.

Each of these weeks will end with a **short written assignment** that covers the main learning goals of the three associated lectures. These assignments are part of Check6-1 and Check6-2.

Hw6 is a Guided Project

Hw6 is organized differently from the past assignments. In this homework, you will spend three weeks **building a code project** that uses computer science in some domain.

This project will be **heavily guided**, with lots of algorithmic instruction in the writeup. It will also have two check-ins at Check6-1 and Check6-2 before the full project is due in Hw6.

Most importantly – you get to choose which project you complete!

Hw6 Project Options

Each of the five projects implements one of the three applications we'll teach in class.

Battleship is focused on building a game. It uses **simulation**.

Circuit Simulator is focused on implementing circuits. It uses **simulation**.

Language Modeling is focused on identifying patterns in text. It uses **data analysis/machine learning**.

Protein Sequencing is focused on analyzing DNA data. It uses **data analysis**.

Social Media Analytics is focused on analyzing political Twitter and Facebook data. It uses **data analysis**.

Hw6 Schedule

Here are the important deadlines for Hw6:

Friday 11/20 noon – Fill out this form to select which project you plan to do:
https://forms.gle/2cN9za2YJqSWZySv7

Monday 11/23 noon – Check6-1 is due (Hw6 check-in, and written assignment)

Friday 12/04 noon – Check6-2 is due (Hw6 check-in, and written assignment)

Thursday 12/10 noon – Check6-1 and Check6-2 revisions due

Friday 12/11 noon – Hw6 is due (full project, including work from both check-ins).
Note that Hw6 does not have a revision deadline.

Code Organization

Helper Functions

In Hw6 (and in projects you might work on outside of 15-110), the code you write will be bigger than a single function. You'll often need to write many functions that work together to solve a larger problem.

We call a function that solves part of a larger problem this way a **helper function.** By breaking up a large problem into multiple smaller problems, and solving those problems with helper functions, we can make complicated tasks more approachable.

In Hw6, we've broken problems down into helper functions for you. If you work on a separate project, you'll need to do this process on your own. Try to identify **subtasks** that are repeated or are separate from the main goal, and have **one subtask per function**.

Example: Tic-Tac-Toe

Consider the game tic-tac-toe. It seems simple, but it involves multiple parts to play through a whole game.

If you implemented Tic-Tac-Toe in Python using helper functions, the main function might look like the code to the right.

Then you would have to implement `makeNewBoard`, `findWinner`, and `takeTurn` to each solve the intermediate problems.

These functions might need their own helper functions too- for example, `takeTurn` might need a function `isLegalMove` to check if a player's move is allowed.

```
def playGame():
    print("Let's play tic-tac-toe!")
    board = makeNewBoard()
    player1Turn = True
    while findWinner(board) == None:
        if player1Turn:
            takeTurn(board, "X")
        else:
            takeTurn(board, "O")
        player1Turn = not player1Turn
    print("Goodbye!")
```

Multiple Files

When working on an especially large project, you may need to split code across multiple **files** in addition to multiple functions. We did this with the MapReduce example last unit!

Generally, each file has a theme that is shared by all the functions inside of it. Maybe they all relate to the graphical interface of the program; maybe they're all core tools that are used by all the other parts of the program.

You can access the functions inside a file from a different file by using the `import` command – it works on your own files in addition to Python modules! For example, if you have created a collection of tools in the file `tools.py`, to access the function `average(lst)` in that file, include the lines:

```
import tools
example = [ 1, 2, 3, 4 ]
tools.average(example)
```

Code Maintenance

Coding for Real Projects

You'll leave this course with a basic working knowledge of programming, which you may want to apply to your own projects. But if you plan to write code for real projects, you'll need to treat that code like an artifact that others will use. This comes with a new set of recommendations and rules for coding.

We'll focus on four main rules: **comment, test, attribute,** and **use good style**.

Rule 1: Comment Your Code

Up until now, we've primarily used comments to give instructions on assignments, and maybe to comment out non-working code.

In real projects, comments should be used to add **documentation** to your code. This makes it possible for other people to understand what your code does by scanning the comments, instead of trying to parse the code.

As a starting point, it's good practice to have a big comment at the top of every file, and smaller comments on every function that describe what they do.

Rule 2: Write Tests for Your Code

In this class, we've provided test cases for you to check your code. In real life, you'll need to write tests on your own, to make sure your code does what it's supposed to.

Test cases are primarily useful for **refactoring and updating** – that is, making sure you don't break your code if you change it later on. Refactoring is changing the structure of code without changing its purpose, e.g., you might move some functions to a different file, or change the order of inputs that a function accepts.

Make sure to create test cases that cover all the core functions in your code, including helper functions!

Rule 3: Attribute Code to Its Author

You'll sometimes find a useful bit of code in a StackOverflow post or a GitHub project that you want to use in your own project.

Whenever you copy code from online, make sure to **cite it** the same way you would cite a paragraph of text in an essay. You can do this by putting a comment above the copied code that includes a link to the URL you got the code from.

This serves two purposes. First – it gives credit to the individual who originally wrote the code. Second – if you run into a problem with the code later on, you'll be able to look back to the original source to find a solution.

Note: policies around copying code change when you're working on a commercial product. Read the fine print if you're planning to sell your code!

Coding with Other People

You might occasionally need to write code with another person, or with a team of people. When this happens, you may need to use **style guides** for the code you write together.

Why do we need style guides? Let's look at an example of code without and with good style...

What does the following program do?

```
def f(a):
    f = False
    if a<2 :    return f
    for i in range (2,a) :
        if (((a)%(i))==
        0)==True:
            return not f
    return f
```

```
def isPrime(num):
    if num < 2:
        return False
    for factor in range(2, num):
        if num % factor == 0:
            return True
    return False
```

Rule 4: Code with Style

A **style guide** for coding is like a style guide for writing – it's a set of rules that describe how you should format the code that you write.

Style guides let you standardize format across multiple people, so that everyone can easily read and modify each other's code.

Python's official style guide is PEP 8, but different organizations and companies may have their own style guides. Google's Python style guide is here.

Real Life Implications

Why does all of this matter? Computer science is a very open-source field, and people share and use each other's code all the time.

However, you can't write code once, share it with others, and then be done with it. Code lives in an environment that is constantly changing – languages evolve, new OS versions are released, and expectations change. **Modules regularly need to be updated to fix bugs and respond to language changes**.

Security Concerns with Legacy Code

Many companies (and governments) rely on old code systems that have not been updated in decades. This makes it difficult to upgrade systems, and also leaves organizations open to security threats.

A recent analysis of US government IT systems showed the government spends 80% of its IT budget on legacy code maintenance, and that ten different systems across different agencies pose critical security risks.

Of those systems, three have been used for over 30 years!

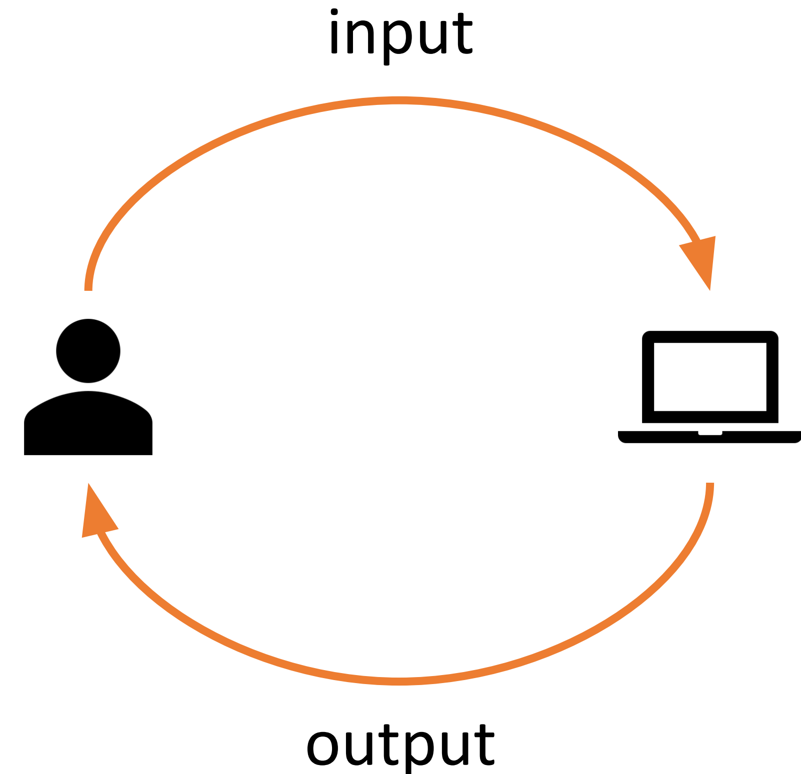
Another example: states with unemployment systems implemented in COBOL were desperate for COBOL programmers this past summer.

User Input

Input-Output Cycle

When you use a program (like your internet browser, or Pyzo), you're communicating with the computer. You give the computer **input** on what you want to do, and it produces **output** based on your requests.

When we write programs, we can capture input directly from the user. When combined with output (printed strings), we can make programs that interact with a user!



Getting Input from the User

Up until now, we've written programs that get their input solely from the function arguments we provide. Alternatively, we can write programs that ask the user to enter information while the program is running.

The built-in function `input(msg)` displays a message in the interpreter, lets the user type a response in the interpreter, then **returns the response** as a string when the user presses enter.

```
name = input("Enter your name: ")
print("Hello, " + name + "!")
```

`input()` Returns a String

Note: users sometimes enter unexpected whitespace at the beginning or end of a response. The built-in function `s.strip()` may prove useful!

`input()` will **always** return a string. If we want to use a user's response as a number, we need to use type-casting to change it.

```
age = int(input("Enter your age: "))
print("You'll be", age + 1, "next year")
```

Handle User Errors with Try-Except Statements

What happens if we ask the user to enter a number and try to convert their text to a number, but they enter a non-number instead?

The code will throw a `TypeError` when it tries to convert the text to an int. This is not great, because users get frustrated if the program crashes every time they make a mistake.

In order to make a program robust against human errors, we can use a **try-except** control structure to recover from such errors.

Try-Except Statements

A **Try-Except** statement looks like this:

```
try:
        <try-block>
except:
        <what to do if the try code throws an error>
```

This works a bit like an if-else statement. Start in the `try` block. the code in the `try` block runs correctly, the `except` block is skipped. Alternatively, if Python encounters a runtime error in the `try` block, it immediately exits that block and jumps to the beginning of the `except` block.

Example: Inputting a Number

Let's try our age-entering program again, this time with error handling.

```
try:
    age = int(input("Enter your age:"))
    print("You'll be", age + 1, "next year")
except:
    print("That's not a real age!")
```

Note that the first print statement does not run if the user enters a non-number into the input.

Activity: Write error-catching code

You do: write a short snippet of code that asks the user to enter two numbers (with two separate `input()` calls), then prints the result of multiplying those two numbers. If one of the inputs isn't a number, print a short error message using an `except` block.

Test your code by trying good inputs (two inputs) and bad inputs of different kinds.

User Input Loops Ensure Correct Inputs

Sometimes you might need to try several times to get the user to input a valid option into the program.

When you need to get a real input from a user, use a **loop** to continue asking them for input until they get it right.

What's the loop control variable? Either the variable you're setting based on the user's entry, or a Boolean you set up specially.

Example: Entering y/n

For example, let's write a simple program that requires the user to respond with either y (yes) or n (no).

```
valid = False
while not valid:
    answer = input("Do you like ice cream? [y/n]:")
    if answer == "y" or answer == "n":
        valid = True
    else:
        print("Seriously, answer the question.")
if answer == "y":
    print("Me too!")
else:
    print("Lactose intolerance sucks :(")
```

Python Modules

Python Modules

The Python programming language comes with a large set of build-in functions that cover a range of different purposes. However, it would take too long to load all these functions every time we want to run a program.

Python organizes its different functions into **modules**. When you run Python, it loads only a small set of functions from the built-in module. To use any other modules, you must **import** them.

Built-in Modules

We've already used a few core modules for homework assignments - mainly `math` and `tkinter`.

For a full list of python libraries, look here: https://docs.python.org/3/library/index.html

Let's briefly introduce two other useful modules – `datetime` and `os`. We'll learn about other modules later in this unit as well!

`datetime` Module

If you're working on a dataset that includes timestamps, the datetime module is useful for parsing information out of the timestamp.

There are functions that let you get the day, month, hour, minute, or whatever else you might want out of a timestamp. You can also convert timestamps between different formats (like `"mm/dd/yy"` to `"dd-mm-yyyy"`).

Use `datetime.datetime.now()` to get the timestamp at the moment the line of code runs. This is useful when you're generating log files.

os module

The os module lets you directly interact with your computer's **operating system**. The operating system is the part of the computer that manages the low-level operations, such as deciding which program gets to run on the processor and deciding where data is stored in memory.

You can use the os module to modify files on your computer. The following functions are especially useful:

```
os.listdir(path) # returns a list of files in the directory
os.path.exists(path) # returns True if the given path exists
os.rename(a, b) # changes file a's name to b
os.remove(path) # deletes the file.
# WARNING: deleting via program is permanent! Not in Trash.
```

External Modules

There are many other libraries that have been built by developers outside of the core Python team to add additional functionality to the language. These modules don't come as part of the Python language, but can be added in. We call these **external modules**.

In order to use an external module, you must first **install** it on your machine. To install, you'll need to download the files from the internet to your computer, then integrate them with the main Python library so that the language knows where the module is located.

Finding Useful Modules

One of the main strengths of Python as a language is that there are thousands of external modules available, which means that you can start many projects based on work others have done instead of starting from scratch.

You can find a list of popular modules here: wiki.python.org/moin/UsefulModules

And a more complete list of pip-installable modules here: pypi.org

We'll go over a few of the most popular modules among CMU students in the next few slides.

`pip`

It is usually possible to install modules manually, but this process can be a major pain. Luckily, Python also gives us a streamlined approach for installing modules – the **`pip` module**! This feature can locate modules that are indexed in PyPI, the Python Package Index (a list of commonly-used modules), download them, and attempt to install them.

Traditionally, programmers run `pip` from the **terminal**. This is a command interface that lets you make changes directly to your computer. But in this class, we'll just run `pip` in Pyzo.

Running pip

To run `pip` in Pyzo, use this command in the interpreter

```
pip install module-name
```

This will identify the module and your version of Python, then start the download and installation process. It may run into a **dependency error** if the module needs a second module to already be installed – in general, installing that module and then running `pip` again will fix the problem.

Note: you will not be able to run `pip` on CMU cluster machines, as these have restricted permissions. You may need to log into your main account on personal machines to run it.

Using an Installed Module

Once you've successfully installed a module, you should be able to put

```
import module-name
```

at the top of a Python file, and it will load the module the same way it would load a built-in library.

Note: this may fail if you have multiple versions of Python installed on your machine and you install in the terminal. Make sure to use the `pip` associated with the version of Python you're using in your editor. You can check your editor's version in Pyzo with Shell > Edit Shell Configurations (check the value in exe), then call `pip` using

```
pythonversion-number -m pip install module-name
```

Learning how a Module Works

Once a new module is installed, you're still left with a major question: how do you use it?

This varies by module, but the best answer is to **read the documentation**. Most external modules have official documentation or APIs that describe which functions exist and how to use the module.

It can also be helpful to search online for other projects that have used the same module, to find examples of how to set it up. Many people have written helpful tutorials online for this exact purpose.

Two standard resources for finding help are StackOverflow, a site where people can ask questions about code and get answers from other developers, and GitHub, a site where people post open-source projects for others to use and contribute to.

Remember to cite any code you get from online!

Useful Modules

Useful Modules

This final section describes an array of modules that other Python programmers (and CMU students!) find useful.

You are not responsible for understanding all of these modules. Think of this more as a resource to consult if you decide you *want* to use any of them later on.

SciPy Collection

SciPy is a group of modules that support advanced mathematical and scientific operations. It can handle large calculations that might take the default Python operations too long to compute. We'll use this a little bit in the Data Analysis lectures.

The group includes NumPy (which focuses on math), SciPy (science), pandas (data analysis), and Matplotlib (plotting of charts and graphs). These can be used separately or as a group. Each needs to be installed separately, but can be installed directly with `pip install name`

Website: https://www.scipy.org/

SciPy Collection Example

We'll show how to use Matplotlib in the Data Analysis II lecture. To learn about pandas, watch the following video: http://www.cs.cmu.edu/~15110/hw/hw6_pandas.mp4

Here's a brief demo of running a t-test with Numpy and Scipy:

```
# Run a T-test on two random sets of data
import numpy, scipy
from scipy import stats
vals1 = numpy.random.random(1000)
vals2 = numpy.random.random(1000)
result = stats.ttest_ind(vals1, vals2)
print(result.pvalue)
```

`scikit-learn`

`scikit-learn` is a module that supports a large set of machine learning algorithms in Python. If you want to dabble in machine learning or artificial intelligence, this is a good place to start. Note that you'll still need to provide a starting dataset to get any algorithm to work.

Website: https://scikit-learn.org/stable/

Install:

```
pip install scikit-learn
```

scikit-learn Example

```
# Learn a decision tree from a random set of two-number data points
# that predict a third number
import numpy
import sklearn
from sklearn import tree
import matplotlib.pyplot as plt

trainingX = [ numpy.random.random(2) for i in range(1000) ]
trainingY = numpy.random.random(1000)

regr = tree.DecisionTreeRegressor(max_depth=2)
regr.fit(trainingX, trainingY)

plt.figure()
tree.plot_tree(regr)
plt.show()
```

`nltk`

`nltk`, the Natural Language Toolkit, assists with natural language processing for machine learning purposes. This is useful whenever you're working with a corpus of written texts.

Website: https://www.nltk.org/

Install:

```
pip install nltk
```

nltk Example

```
# Identify the nouns in a document
import nltk
document = "insert example text here"
result = []
words = nltk.word_tokenize(document)
tags = nltk.pos_tag(words)
for tup in tags:
    (word, type) = tup
    if (word.lower() not in result) and (type == 'NN' or type == 'NNS'):
        result.append(word.lower())
result.sort()
print(result)
```

Beautiful Soup

Beautiful Soup is a module that supports webscraping and HTML parsing. This is useful if you want to gather data from online for use in an application.

Website: https://www.crummy.com/software/BeautifulSoup/

Install:

```
pip install beautifulsoup4
```

Parse HTML as Tags

HTML organizes content on a page using **tags**, like this:

```
<tag attribute="value">
  <subtag> Some content for the subtag </subtag>
</tag>
```

To parse a website, you need to look for a certain type of tag in the file.

Beautiful Soup Example

```
import requests
from bs4 import BeautifulSoup

page = requests.get("https://www.cs.cmu.edu/~110/schedule.html")
soup = BeautifulSoup(page.content, 'html.parser')
for link in soup.find_all('a'):
    url = link["href"]
    if "slides/" in url and ".pdf" in url:
        print(link["href"])
```

PyAudio

PyAudio makes it possible to analyze, create, and play audio files. This module requires some complex pre-existing software, including the language C++; if you get an error message while installing, read it carefully to see how to make the installation work.

Website: https://people.csail.mit.edu/hubert/pyaudio/

Install:

```
pip install pyaudio
```

Note that there are many other audio modules available as well; you can find a list: https://wiki.python.org/moin/Audio

PyAudio Example

```
import pyaudio, math
# Make the sound data
bitrate, freq = 64000, 130.815 # C3 frequency
dataFrames = [""] * 5
for octave in range(5):
    mult = 2*math.pi*(freq*(octave+1))
    for frame in range(bitrate):
        dataFrames[octave] += chr(int(math.sin(mult * frame / bitrate)*127+128))
# Play the sounds!
p = pyaudio.PyAudio()
stream = p.open(format=32, channels = 1, rate = bitrate, output = True)
for frame in dataFrames:
    stream.write(frame)
# Close the stream
stream.stop_stream()
stream.close()
p.terminate()
```

Django

Django is a module that lets you build interactive websites using Python. This involves setting up a **frontend** (the part of a website that the user sees while browsing) and a **backend** (the part of a website that processes requests and does the actual work).

Website: https://www.djangoproject.com/

Install:

```
pip install django
```

Fun Modules

PIL: Python Imaging Library

PIL is a lightweight and easy-to-install module that lets `tkinter` interact with images other than .gif and .ppm files. It also includes functions that support basic image manipulation.

Website: http://www.pythonware.com/products/pil/

Since the main PIL installation is no longer maintained, most programmers use an offshoot called Pillow instead.

Website: https://pypi.org/project/Pillow/2.2.1/

Install:

```
pip install pillow
```

Pygame

Pygame is, like tkinter, a library that lets you make graphical applications. However, Pygame is specifically designed to create games. It has better support for sprites and collision detection than `tkinter`.

Website: https://www.pygame.org/news

Install:

```
pip install pygame
```

Panda3D

Panda3D is a module that supports 3D rendering and animation. Like `pyaudio`, it can be very complicated to install and use, but it is still much easier than trying to create 3D animation in a 2D system.

Website: https://www.panda3d.org/manual/

Install:

```
pip install Panda3D
```

Learning Goals

- Implement **helper functions** in code to break up large problems into solvable subtasks
- Recognize the four core rules of **code maintenance**
- Use the **input** command and **try/except** structures to handle direct user input in code
- Learn how to install and use **external modules**
- **Feedback:** https://bit.ly/110-feedback