Programming Languages and Software Development

- Programming Languages
- Program Structure
- Problem Solving
- Object-Oriented Programming
- Reading for this class: *Dawson*, Chapter 1/2

What is a program?

- It consists of two components:
  - Data (numbers, characters, true/false)
  - Steps
- A program goes through a number of steps with pieces of data to achieve a result:
  - Printing text to screen
  - Collecting information
  - Performing calculations
- Example: Long Division

Programming Languages

- Computer programmers write programs for computers using one or more programming languages
- Some languages are better for one type of program or one style of user interface than for others
- You may have heard of some programming languages: Basic, Lisp, C/C++, Java, Python, Assembly Language, and Others

"Hello, World" Versions

- **Java:**

```
public class Hello {
public static void main(String[] args) {
System.out.println("Hello World");
}
}
```

- **Basic:**
```
10 PRINT "HELLO WORLD"
```
- **Fortran:**
```
PROGRAM HELLOWORLD
10 FORMAT (1X,11HHELLO WORLD)
WRITE(6,10)
END "HELLO WORLD"
```
- **Python:**
```
print ("Hello World")
```

- **C:**

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
 printf("Hello, world\n");
 return EXIT_SUCCESS;
}
```

- **Scheme:**

```
(display "Hello, World!")
(newline)
```

Source: http://c2.com/cgi/wiki?HelloWorldInManyProgrammingLanguages

Programming Languages

- A *programming language* specifies the words and symbols that we can use to write a program
- A programming language employs a set of rules that dictate how the words and symbols can be put together to form valid *program statements*
- A programming language has both *syntax* and *semantics*

Syntax and Semantics

- The *syntax rules* of a language define **how we can put together** symbols, reserved words, and identifiers to make a valid program
- The *semantics* of a program statement define what that statement **means** (its purpose or role in a program)
- A program that is syntactically correct is not necessarily logically (semantically) correct
- A program will always do what we **tell** it to do, not what we **meant** to tell it to do

Syntax vs. Semantics

- **Everyday Language:**
  - –Incorrect syntax: Ball the red is color the
  - –Incorrect semantics: The ball is the color three
  - –Alternative (not incorrect) syntax:

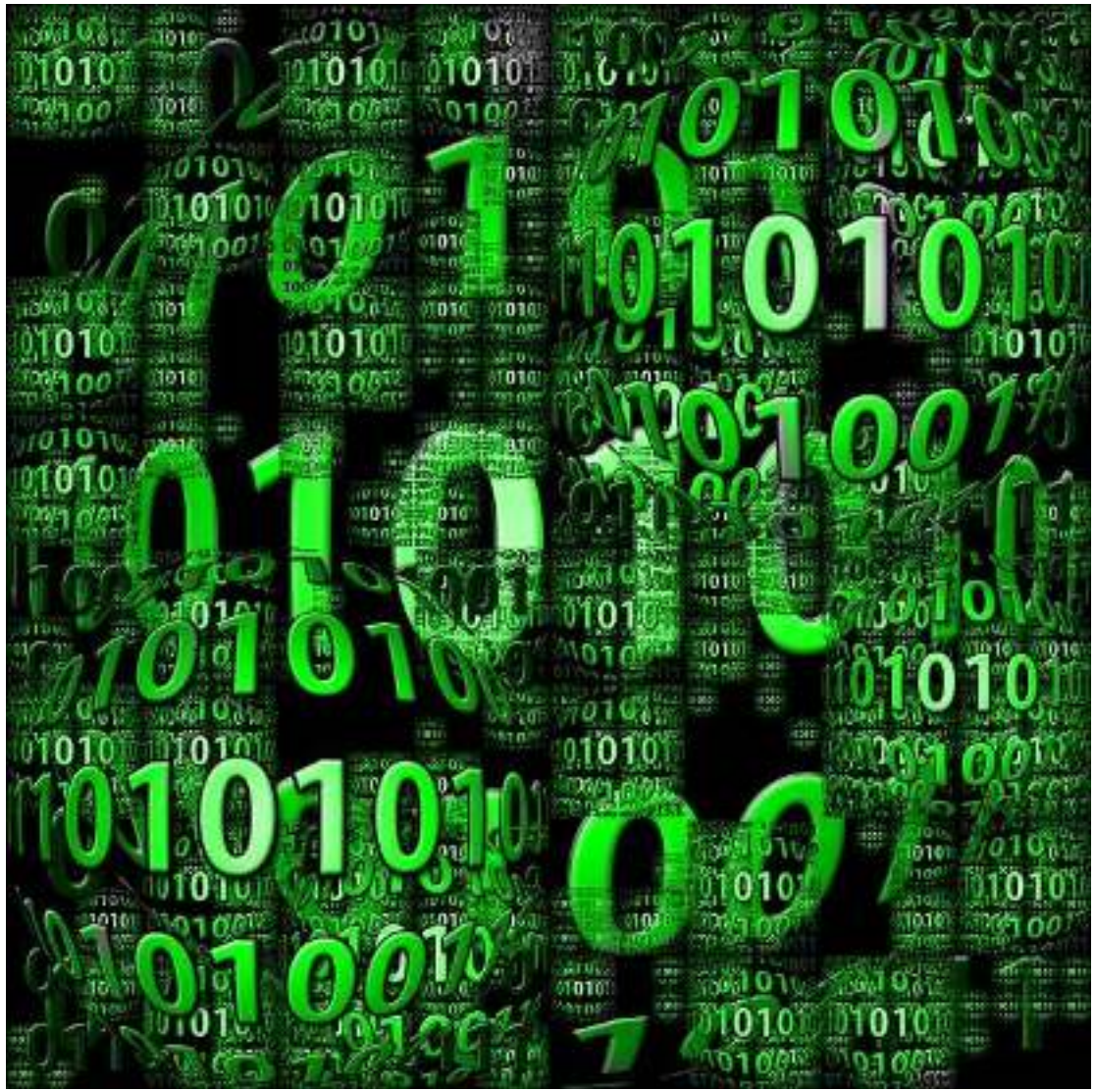
    "Finish your training, you must" – Yoda
- **Programming (ex. Java):**
  - –Correct: `int i = 34;`
  - –Incorrect semantics: `int i = "foobar";`
  - –Incorrect syntax: `i = 34`

However, the last example **would be correct in another language – e.g., Python*

Language Levels

- There are four programming language levels:
  - –machine language
  - –assembly language
  - –high-level language
  - –fourth-generation language
- Each type of CPU has its own specific *machine language*
- The other levels were created to make it easier for a human being to read and write programs

Machine Language



http://pixabay.com/static/uploads/photo/2013/11/24/10/20/ball-216837_640.jpg

Programming Languages

- Program instructions must be translated into *machine* instructions before the program can be executed
- Many programming languages are *compiled*, which means the code you write is translated into a sequence of machine instructions.
- A *compiler* is a program which takes *source code* as its **input** and returns the translated code as its **output**.
- Examples include C/C++ and Java. Compiled programs often run faster because they are directly in machine language.

Interpreted Languages

- Many programming languages, such as Python, are called "interpreted languages", which means that the source code is directly read and executed when you run the program.
- In addition, interpreted languages often have a program called a "shell", in which you can enter and execute instructions individually.
- Because the interpreter reads source code each time, execution may be slower, as compared to a compiled program. However, this is less of a problem as computational resources (memory, processor speed, etc.) become more available.

Python Program Structure

- In the Python programming language:
  - A program is made up of one or more instructions, or *statements*, which perform operations upon various pieces of data
  - Data may be stored in *variables*
  - Related groups of statements may be organized into *methods*
  - Related variables and methods may be organized into larger units, such as *classes* and *modules*
- These terms will be explored in some detail throughout the course

Basic Definitions

- ***Statement:*** A piece of code representing a complete step in a program
- ***Variable*:** A named space in program memory for storing a piece of data.
- ***Method*:** A named set of instructions that acts upon supplied data in order to accomplish some goal
- ***Module*:** A body of pre-written code that you can incorporate into a Python program
- ***Class*:** A way of organizing variables and methods, usually for modeling a real-life entity

Python Program Structure

- For now, each program of ours will consist of a single file, which will be entitled `file_name.py`
- `.py` is the extension for Python files
- The file will contain the list of instructions for program execution
- Writing these program files so that the program runs in the intended manner will require ***meticulous*** attention to detail.
- Later on, we will begin to work with programs that consist of *multiple* Python files

Identifiers

- *Identifiers* are the words a programmer uses in a program
- Rules:
  - Can be made up of letters, digits, and the underscore character (_)
  - Identifiers cannot begin with a digit
  - *Case sensitive* - `Total` , `total` , and `TOTAL` are different identifiers
- By convention, programmers use different case styles for different types of identifiers:
  - *title case* for class names – `BankAccount`
  - *lower* case for variable names – `name` , `current_temp` , `speed_limit`
  - *upper case* for constants – `MAXIMUM`

Identifiers

- Sometimes we choose identifiers ourselves when writing a program (such as `BankAccount`)
- Sometimes we are using another programmer's code, so we use the identifiers that he or she chose (such as `print`)
- Often we use special identifiers called *reserved words* that already have a predefined meaning in the language
- A reserved word cannot be used in any other way

Reserved Words

- The Python reserved words:

```
False      class      finally    is         return
None       continue   for        lambda     try
True       def        from       nonlocal   while
and        del        global     not        with
as         elif       if         or         yield
assert     else       import     pass
break      except     in         raise
```

Comments

- Comments in a program are called *inline documentation*
- They should be included to explain the purpose of the program and describe processing steps
- They do not affect how a program works
- Python comments can take at least two forms:

```
# this comment runs to the end of the line
```

```
"""
  this comment can run across several
  lines.  It starts with the first trio
  of quotes above and ends another trio
"""
```

```
"""
   This is the Hello code, which might be
   stored in a file called hello.py
"""

# printing "Hello"
print ("Hello")
```

But, don't comment out code you want to keep!

```
print # ("Hello, world!")
```

Error!

White Space

- Spaces, blank lines, and tabs are called ***white space***
- White space is used to separate words and symbols in a program. Extra white space is usually ignored, depending on the language
- A valid program can be formatted many ways
- Programs should be formatted to enhance readability, using consistent indentation
- In Python, in particular, correct use of indentation is necessary in order to indicate organization of code, as we will see soon.

Formatting Poorly

```
a="Nancy"
b=21
c="Biology"
d="Bill"
e=23
f="English"
print(a, "is", b, "years old, and her major is", c, "\n",
d, "is", e, "years old, and his major is", f)
```

It compiles and runs fine, so what's wrong here?

Hard to read (No use of spacing, indentation, tabs)

Meaningless identifiers

No commentary

Truly, a nightmare come true – for the next person who has to maintain this code!

Formatting Well

```
first_student_name = "Nancy"
first_student_age = 21
first_student_major = "Biology"

second_student_name = "Bill"
second_student_age = 23
second_student_major = "English"

print(first_student_name, "is", first_student_age, "years
old, and her major is", first_student_major)
print(second_student_name, "is", second_student_age,
"years old, and his major is", second_student_major)
```

Problem Solving

- The purpose of writing a program is to solve a problem
- Solving a problem consists of multiple activities:
  - Understand the problem
  - Design a solution
  - Consider alternatives and refine the solution
  - Implement the solution
  - Test the solution
- These activities are not purely linear – they overlap and interact (*for example, see the "iterative" diagram from the previous lecture*)

Problem Solving

- The key to designing a solution is *breaking it down* into *manageable pieces*
- When writing software, we design separate pieces that are responsible for certain parts of the solution
- An *object-oriented approach* lends itself to this kind of solution decomposition
- We will dissect our solutions into pieces called ***objects*** and ***classes***

Object-Oriented Programming

- Python is an object-oriented programming language
- As the term implies, an *object* is a fundamental entity in a Python program
- Objects can be used effectively to represent real-world entities
- For instance, an object might represent a bank account
- Each bank account object handles the processing and data management related to that bank account

Objects

- An object has:

  –*state* - descriptive characteristics (variable values)

  –*behaviors* - what it can do (or what can be done to it)

- The **state** of a bank account includes its balance
- The **behaviors** associated with a bank account include the ability to get the balance, make deposits, and make withdrawals
- Note that the behavior of an object might change its state, e.g. making a deposit will increase the balance

Classes

Class (The Conceptual)

- Is like a blueprint for...
- Has *attributes* that...
- Has *methods* that...

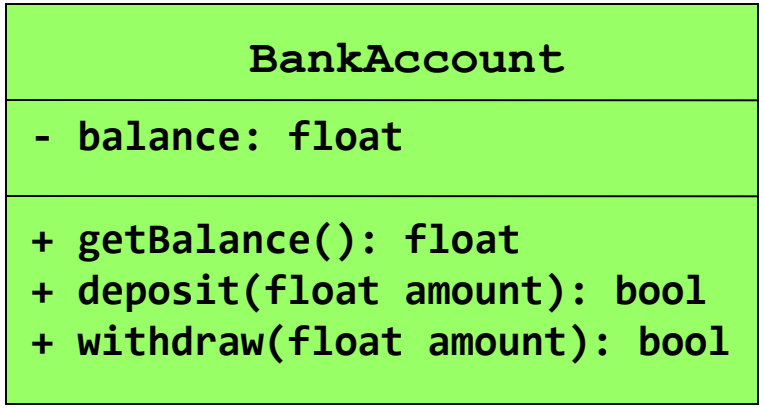
Object (The Concrete)

- An object
- Define the *state* of each object
- Define the *behavior* of each object

- Typically, an advanced program in Python, like in any other language, will make use of multiple classes and modules
- These elements are the building blocks of useful software

Objects and Classes

A Class
(The Concept)



Three objects
(Three Instances
of the Concept)

John's Bank Account
Balance: $5,257.51

Bill's Bank Account
Balance: $1,245,069.89

Mary's Bank Account
Balance: $16,833.27

Multiple objects
of the same class

Printing

- One of the most basic steps in a simple command-line application is printing text to the screen
- There are a number of variations on this step, some simpler and some more complex
- As you move along, you will find it helpful to have a variety of printing techniques available for your use
- We will go over four now…

Basic print

- The simplest approach is the following:

```
print ("some text")
```

- It involves these components:
  - The method name: `print`
  - The pair of parentheses: `()`
  - The contents of the parentheses – in this case: `"some text"`
- It will print all the text and then skip to the next line – like pressing `Enter` while typing

Printing a list

- There is a variation of the `print` method, which accepts a *list* of items (separated by *commas*) and prints them to a line, separated by spaces:

```
print ("Hello, my name is", "Bob")
```

- It prints: ***Hello, my name is Bob***
- You may find this helpful in situations where *parts of the printed information vary*
- Example:

```
temp = 97
print ("It is", temp, "degrees today!")
```

Printing with a custom line ending

- The default behavior of the **`print`** method is to print the specified information and skip to the next line (*i.e., it ends with a new line*)
- However, sometimes you may not want this, in which case you can use a custom line ending, like so:

```
print ("Hello, my name is", end=": ")
print ("Bob")
```

- This will produce the following output:

```
Hello, my name is: Bob
```

Printing multiline

- Just as you can use triple-quotes (`"""`) to create a multiline comment, you can also use them to create a *multiline string*, which will print **just** the way you type it:

```
print(
"""
Happy
    Birthday
        to
            You!
"""
)
```

Will print:

```
Happy
    Birthday
        to
            You!
```