Lecture 1: Introduction to Computer Programming Course - CS1010

DEPARTMENT OF COMPUTER SCIENCE | 08/30/2019

Rensselaer

Welcome to CS1010!

- **Meet the Instructor(s)**:

Course Instructor: Dr. Uzma Mushtaque, PhD	mushtu@rpi.edu
Office Location: Amos Eaton 111	
Office Hours: Tuesday: 12:00 pm to 1:30 pm Friday: 12:00 pm to 1:30 pm Or by appointment.	
Course (Teaching Assistant)TA: Jessup Barrueco	barruj@rpi.edu
Office Location:	Lally lab
Office Hours:	Wednesday 3-4pm
Course Mentors:	
Shreya Barua	baruas@rpi.edu
William Asai	asaiw@rpi.edu
Patrick Berne	bernep@rpi.edu

Structure of Lectures/Class Meeting

- Tuesday
  - Introduction to a new topic.
  - Mostly Lecture and Discussions
  - Few in–class exercises
  - Might have a 10 minutes break
- Friday
  - Continue the Topic from Monday
  - Mostly in-class exercises to be submitted in class
  - Some part can be lecture and/or discussion

Before Starting

- Please have a look at the syllabus
- Be aware of the course schedule
    - Home-works
    - Exams
    - Office -hours
- There is a recommended textbook for the course (not required): *Practical Programming: An Introduction to Computer Science Using Python by Campbell, Gries, and Montojo* (2nd Edition)

Course Assessment Measures

Assessment	Number
Exam/Tests	2-3
Homework	7-9
In class exercise	16-18
Final Exam	1

- Lecture Exercises: 5%,
- Homework: 40%,
- Tests: 35%,
- Final: 20% .

Software Requirements

- We will use Python as the Programming Language for this course.
- Install Python 3.6
- Use Spyder as the IDE (Integrated Development Environment)
- We will download and Install Miniconda in Class next week.
  - Please bring your computers to class on Tuesday.

Resources

- Use Submitty to submit your homework and other coding assignments
- https://submitty.cs.rpi.edu/f19/csci1010

- Check the course website for all the material
- To go to the course website:
    - log onto Submitty
    - Click the ‘Course home’ link on the top left
    - OR Directly go to: https://www.cs.rpi.edu/~mushtu/CS1010/index.html

Course Goals

Ultimately, to give give you the acumen required to solve everyday problems with code.

To convince, motivate and inspire you to see your problems as solvable in an objective manner, with computer science.

Show you how computers work and that programming is fun.

77%

percentage of jobs in the US requiring technology skills by 2025

Review

Computers Don’t Byte!

What is a computer?

Noun: “an electronic device for storing and processing data, typically in binary form, according to instructions given to it in a variable program.”

How many computers do you have on you at this given moment.

- Laptop, Watch, Phone, Calculator

Parts of a Computer System:

- Hardware: Electronic Devices
- Software: Instructions and Computer Programs

That’s great, but how does a computer use these components to “think”?

Computer Language

- Digital devices have two stable states, which are referred to as zero and one by convention

- Binary Language: Data and instructions (numbers, characters, strings, etc.) are encoded as binary numbers - a series of bits (one or more bytes made up of zeros and ones)

Computer Language (cont.)

- Encoding and decoding of data into binary is performed automatically by the system based on the encoding scheme
- Encoding schemes
    - Numeric Data: Encoded as binary numbers
    - Non-Numeric Data: Encoded as binary numbers using representative code
        - ASCII – 1 byte per character
        - Unicode – 2 bytes per character

Programming Languages

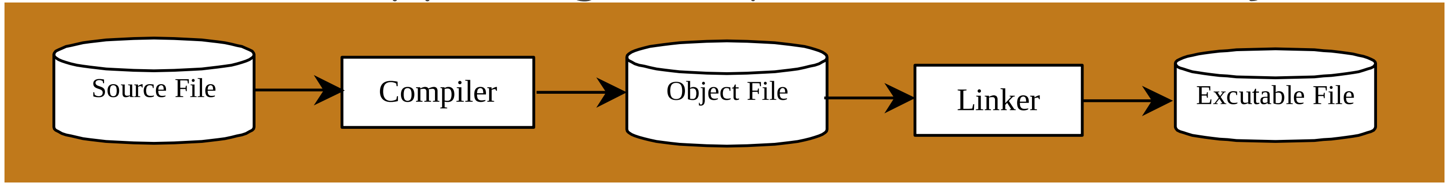
- Computers can not use human languages, and programming in the binary language of computers is a very difficult, tedious process
- Therefore, most programs are written using a programming language and are converted to the binary language used by the computer
- Three major categories of prog languages:
  - Machine Language
  - Assembly Language
  - High level Language

High Level Languages

- English-like and easy to learn and program
- Common mathematical notation
    - Total Cost = Price + Tax;
    - area = 5 * 5 * 3.1415;
- Java, C, C++, VISUAL BASIC, Python

Compiling Source Code

A program written in a high-level language is called a *source program* (*or source code*). Since a computer cannot understand a source program. Program called a *compiler* is used to translate the source program into a machine language program called an *object program*. The object program is often then linked with other supporting library code before the object



Programming

- Programming – the creation of an ordered set of instructions to solve a problem with a computer.
- Different programs will just use these instructions in different orders and combinations.
- The most valuable part of learning to program is learning how to think about arranging the sequence of instructions to solve the problem or carry out the task

Programming Fundamentals: Putting the Instructions Together

- Sequential Processing: A List of Instructions
- Conditional Execution : Ifs
- Repetition: Looping / Repeating
- Stepwise Refinement / Top-Down Design : Breaking Things into Smaller Pieces
- Calling Methods / Functions / Procedures / Subroutines
    - Calling a segment of code located elsewhere
    - Reuse of previously coded code segment

Methods of Programming

- Procedural
  - Defining set of steps to transform inputs into outputs
  - Translating steps into code
  - Constructed as a set of procedures
  - Each procedure is a set of instructions
- Object-Oriented
  - Defining/utilizing objects to represent real-world entities that work together to solve problem
  - Basic O-O Programming Components: Class, Object, Properties, Methods

Problem Solving

- The process of defining a problem, searching for relevant information and resources about the problem, and of discovering, designing, and evaluating the solutions for further opportunities. Includes:
  - Finding an Answer to a Question
  - Figuring out how to Perform a Task
  - Figure out how to Make Things Work

4 Steps of Problem Solving (Polya's)

- U – Understand the Problem
- D – Devise a Good Plan to Solve
- I – Implement the Plan
- E – Evaluate the Solution

Example:
Solving Math Word Problem

- Read the Problem: Understand the description of problem or scenario, identifying the knowns and unknowns
- Decide how to go about solving the problem: Determine what steps need to be taken to reach the solution
- Solve the Problem: Write the solution
- Test the Answer: Make sure the answer is correct

Solving Computing Problems

- In general, when we solve a computing problem we are taking some **input**s, **process**ing (performing some actions on) the inputs, and then **output**ting the solution or results.
- This is the classic view of computer programming – computation as calculation
- Polya’s steps (UDIE) can be very effective when applied to solving computing problems

Applying Polya’s Problem Solving to Programming

Step 1 - Understand the Problem

- What is the Problem to be solved? What is the unknown? What is the condition? What is the data? What is needed to solve the problem? What actions need to take place?
- Identify the **inputs** and **outputs**
- Identify the **processes** needed to produce the **outputs** from the given **inputs**
- Draw a figure. Introduce suitable notation.
- Isolate Principle parts of the problem.

Applying Polya's Problem Solving to Programming
Step 2 - Devise a Plan

- Find connections between the knowns and unknowns.
- Simplify: Break the problem into smaller sub-problems
- Design a solution
- Make a plan or list of actions to implement the solution
    - Algorithm / Flowchart / Psuedocode

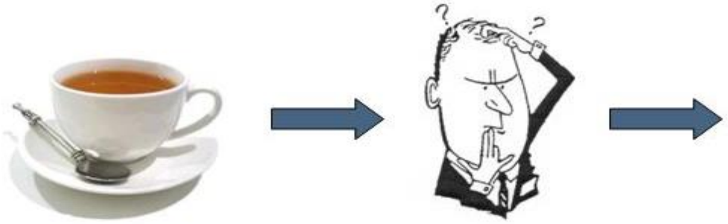
Applying Polya’s Problem Solving to Programming

Step 2 - Devise a Plan (cont.)

- Algorithm
  - A FINITE set of clear, executable steps that will eventually terminate to produce the desired outcome
  - Logical design used to solve problems – usually a list of actions required to perform task
- Pseudocode
  - Written like program code but more “English Like” and doesn’t have to conform to language syntax
- Flowchart
  - Diagram that visually represents the steps to be performed to arrive at solution.

What is an algorithm? (Plan Continued)

- Algorithm = how you go about solving a puzzle; your solution to a task
- Example: how do you make a cup of tea?



1. Boil kettle
2. Put tea in cup
3. Pour boiling water into cup

- In programming, you should plan your algorithm before you start coding

Applying Polya's Problem Solving to Programming

Step 3 - Implement the Plan

- Implement in a Programming Language
- Carry out the plan checking the preliminary results at each step.
- Code A Little Test A lot

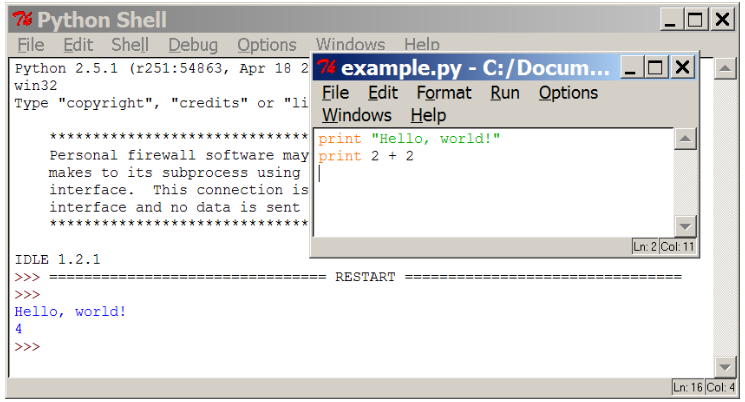
Applying Polya's Problem Solving to Programming

Step 4 - Evaluate the Solution

- Run the Code
- Check results repeatedly and thoroughly
    - Use numerous test cases or data sets
    - Use highly varied test case, including expected as well as and unexpected cases
- Look for new solutions
    - Is there a better, easier, or more efficient solution
- Can other problems be solved using these techniques?

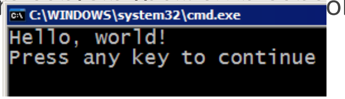
Programming basics

- **code** or **source code**: The sequence of instructions in a program.
- **syntax**: The set of legal structures and commands that can be used in a particular programming language.
- **output**: The messages printed to the user by a program.
- **console**: The text box onto which output is printed.
    - Some source code editors pop up the console as an external window, and other[illegible]ole window.



```
print "Hello, world!"
print 2 + 2
```

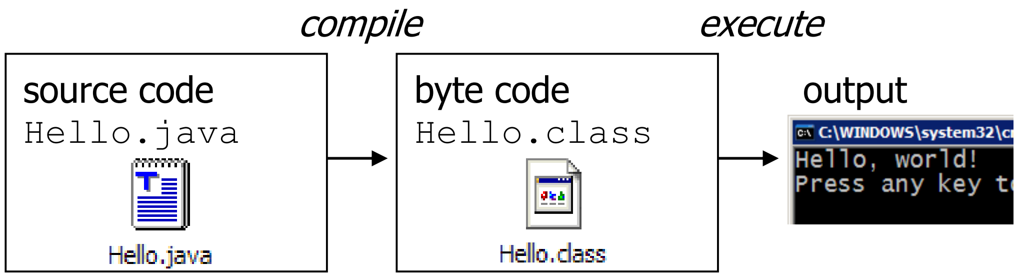
```
Python 2.5.1 (r251:54863, Apr 18 2
win32
Type "copyright", "credits" or "li

    ******************************
    Personal firewall software may
    makes to its subprocess using
    interface.  This connection is
    interface and no data is sent
    ******************************

IDLE 1.2.1
>>> ================================ RESTART ================================
>>>
Hello, world!
4
>>>
```



```
Hello, world!
Press any key to continue
```

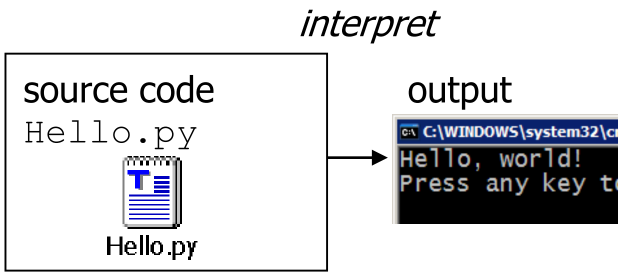
Compiling and interpreting

- Many languages require you to *compile* (translate) your program into a form that the machine understands.



```
Hello, world!
Press any key t
```

- Python is instead directly *interpreted* into machine instructions.



```
Hello, world!
Press any key t
```

Summary

- U - Read the Problem Statement
    - Identify the inputs, outputs, and processes
- D - Decide how to Solve the Problem
    - Create an Algorithm / Flowchart / Psuedocode
- I - Program the Code
    - Implement in Programming Language
- E - Test the Solution
    - Run the Code using numerous, varied test cases

In-Class Exercise

- Write an algorithm to serve as how-to instructions for some relatively simple task or activity. You choose the task, should be 10-15 steps in length. Assume you are writing instructions for someone who has never performed task before.

Getting Started in Programming

What is a programming language?

“a vocabulary and set of grammatical rules for instructing a computer or computing device to perform specific tasks.”

Visual Basic.

```
Sub Main()
   Console.WriteLine ("Hello, World!")
 End Sub
```

Python.

```
def main():
    print("Hello, World!")
```

both have equivalent output

Why Python?

Simple syntax, interpreted language

Many pre written modules

Easy exposure to advanced concepts

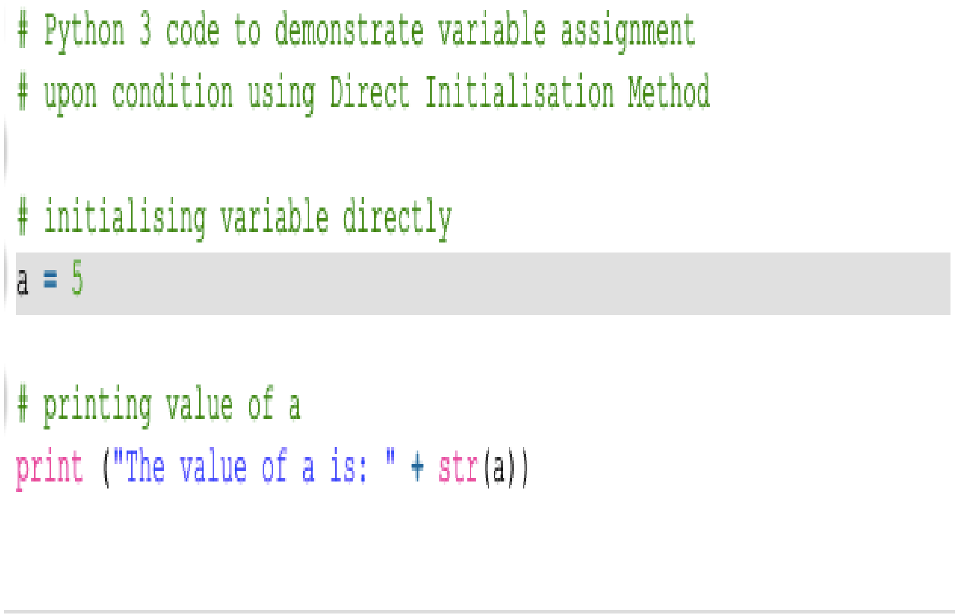
Widely used in many industries

High Level Programming Language

In Python, there is no need to take care about low-level details such as managing the memory used by the program.

No need to declare anything. An assignment statement binds a name to an object, and the object can be of any type.

No type casting required when using container objects

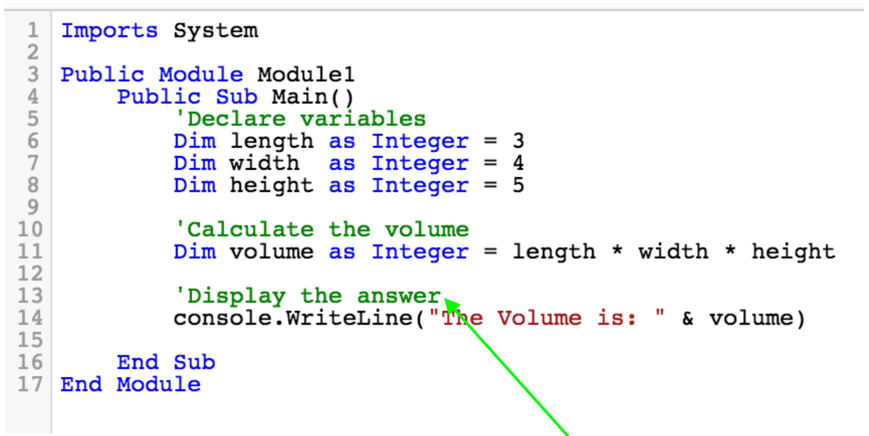


```
# Python 3 code to demonstrate variable assignment
# upon condition using Direct Initialisation Method

# initialising variable directly
a = 5

# printing value of a
print ("The value of a is: " + str(a))
```

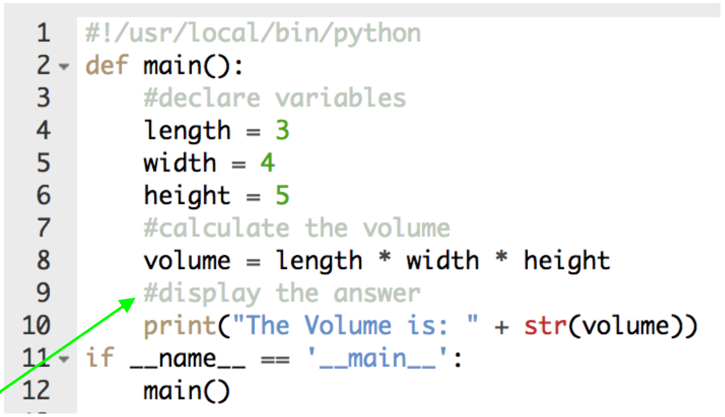
Programming Language Example

Visual Basic: https://dotnetfiddle.net/lelkjT

Python: https://codeenv.com/env/run/R1yVqR/



```
Imports System

Public Module Module1
    Public Sub Main()
        'Declare variables
        Dim length as Integer = 3
        Dim width  as Integer = 4
        Dim height as Integer = 5

        'Calculate the volume
        Dim volume as Integer = length * width * height

        'Display the answer
        console.WriteLine("The Volume is: " & volume)

    End Sub
End Module
```



```
#!/usr/local/bin/python
def main():
    #declare variables
    length = 3
    width = 4
    height = 5
    #calculate the volume
    volume = length * width * height
    #display the answer
    print("The Volume is: " + str(volume))
if __name__ == '__main__':
    main()
```

These are called comments

Low level programming language

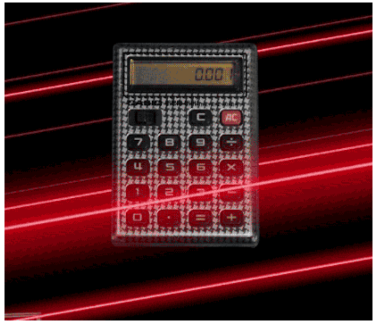
Every bit of logic on your computer is eventually distilled into machine instructions

- Examples below are assembly, machine instructions are their encoded representations (ones and zeros)

Your CPU knows what these encoded instructions are and can execute them.

Examples:

- ADD : add two given numbers and store into a register
- MULT : multiply two numbers and store into a register
- JUMP : go to a specific instruction address and keep executing
- BEQ : if a number is equal to another number jump to an address
- NOOP: literally do nothing



Register 1 Data: 0
Register 2 ADDI R1, R1, 1
Register 3 JUMP R2
Register ...
Register N

High Level Review

Your **code** is made into more primitive instructions your **CPU** can understand. When you run this program, your operating system (the primary program running on your cpu) will allocate system resources (CPU time, RAM et al) , letting the new program operate however you instructed it to.

You will have **data** and **operations**. The composition of your program (your code) will determine what meaningful output the program has.

Next Week

- Install Python on our machines
- Learn to write and run programs
- Python as a calculator

Questions/ Comments?