Intro to Python
Python

• Getting increasingly more common
• Designed to have intuitive and “lightweight” syntax
• In this class, we will be using Python 3.x
  – Python 2.x is still very popular, and the differences between the two are small
  – Python 3.x is better optimized and more regularly improved
  – Python 3.x has slightly different syntax, so be careful (especially when reading stuff on the web)
• Install from https://www.python.org/downloads/
Variables

• A variable is a box that we put some data in
  – Example: \texttt{age = 23}

• When we change the variable, we change what is in the box
  – Example: \texttt{age = age + 1}

• There are different \textbf{data types}
  – \textbf{Integers} – whole numbers (0,1,2,3….-1,-2,-3…)
  – \textbf{Floating-point numbers} – 3.14
  – \textbf{Strings} – “Groups of text” or even single characters (like “a”) are considered strings
  – \textbf{Booleans} – true or false (the basis of all computing)
  – …and more (you can even create your own)

• In Python, you can also change the type of variable (it is \textbf{loosely typed})
  – Example: \texttt{age = “In my 20s”}
  – Not true in Java!
Reading input from the user

• A **function** is a named piece of code that can return a value
• The **input** function is used to read input from the user
  – **input()** just returns a string entered by the user
  – **input(prompt)** displays the “prompt” string, then returns the string entered by the user
  – IDLE example: input.py
• The **input** function always returns a string
• If you want to read a number from the user, use the additional functions **int** or **float**
  – You have to “cast” the string into an int/float
  – More on this later
Doing arithmetic

- Arithmetic is slightly complicated because there are two kinds of numbers, **integers** ("whole numbers") and **floating-point numbers** or **floats** (numbers containing a decimal point).
- Operations are + (add), - (subtract), * (multiplication), two kinds of division, / and //, and % (modulo, or remainder of a division).
  - When you use +, -, *, //, or % on just integers, you get an integer result.
  - // is called **integer division** (works as floor function, i.e., 7//2= 3).
  - All other combinations return a float.
  - IDLE example: arithmetic.py
Using strings

• A string is a sequence of characters enclosed in either single quotes '...' or double quotes "..." (no difference)
  – Triple quotes ‘‘...’’ also work in Python

• Some single characters cannot easily be entered directly into strings, and must be “escaped” (backslashed)
  – \n represents a newline character
  – \t represents a tab character
  – \' represents a single quote (inside a singly-quoted string)
  – \" represents a double quote (inside a doubly-quoted string)
  – Single and double quotes don’t need to be escaped with triple quotes

• Strings can be concatenated (joined) with the + operator
  – IDLE example: strings.py
The **print** function

- **Syntax:** `print(arguments)`
  - The arguments are values, variables, or expressions, separated by commas
  - The arguments are “printed” (displayed on the screen) on a single line, separated by spaces

- Very useful function not only for printing text but also for printing the values of variables
  - More about functions later

- If you’re in IDLE, print is used automatically for the result
Using booleans

• The two **boolean** values are **True** and **False**

• The following comparison operators on numbers will give a boolean result: <, <=, ==, != (“not equal”), >=, >
  
  – 5 > 3 is **True**
  
  – 3 > 5 is **False**
  
  – x == 3 depends on the value of x in the program

• Booleans can also be assigned to variables
  
  – Example: `isXequalTo3 = (x == 3)`

• Booleans can be negated with **not**

• Use **and** and **or** to create more complex booleans
Branching

• Execute different branches of code depending on a condition
• Structure:
  
  ```python
  if condition:
      do something
  else:
      do something else
  ```

• Could have multiple conditions with an `elif`
• IDLE example: `branching.py`
Loops

• A “while loop” executes some statements while some condition is true

  while condition:
    do something

• If condition is true, execute statements and repeat

• IDLE example: guessANumber.py

• A “for loop” iterates over some sequence

  for variable in sequence:
    do something

• variable iterates over all values in sequence

• IDLE example: forLoop.py
Control Statements

• **Control statements** are used to decide whether and how often some other, “controlled” statements are executed
  – If, elif, else statements
  – While, for loops

• For every kind of control statement:
  – The control statement ends in a colon, :
  – **The controlled statements are indented four spaces**
  – In IDLE, pressing the Tab key is the same as typing four spaces
Layout

• Every statement goes on a line by itself (not so in java!)

• Could put an optional semicolon at end of line (semicolon mandatory in java!)

• Indentation is crucial in python!
  – If you don’t indent correctly, your program may do something you don’t expect
Functions

• A function is a named chunk of code that can be executed ("called") by giving its name and whatever arguments it requires
  – The arguments are matched up to the parameters
  – There must be the same number of arguments as there are parameters, and they are matched up by position
  – A function may (or may not) return a value

• Functions declared with a def keyword

• Functions are sometimes called methods
  – More about the difference later

• IDLE example: gcd.py
Programs

• A **program** is code that has been saved to a file
  – In python file has .py extension
  – Can open in IDLE and execute by pressing F5

• A program is executed as it is loaded in, top to bottom.

• It can be either:
  – Just a collection of statements, executed one after the other, or
  – A collection of functions that can be called individually from IDLE, or
  – A collection of functions, plus special code to start the program and call the various functions as needed (more on that later)
Last comments

• Lines starting with # are comments
  – Not executed by the program
  – Note to humans looking at the program
  – Use comments to tell what a program does or acknowledge someone else’s code
  – Do not add irrelevant comments or use comments to explain something that’s obvious (b = a + c #b is assigned the sum of a and c)

• Errors are unavoidable – most of programming is debugging
  – A syntax error is one recognized by the compiler (the thing that gets your program ready to execute) and prevents it from even starting
  – A runtime error is one that causes your program to “crash”
  – A logic error or semantic error is one that causes your program to produce incorrect results
  – A user error is when the user provides invalid input to the program, causing the program to crash or to produce incorrect results
Homework 1 – Lunar Lander

• Due at 11:59pm on Tuesday, Sep. 12

• Fun and simple game
  – Use your fuel wisely so you can land safely on the moon
  – Physics is very simplified so it shouldn’t be difficult to code
  – Mostly testing your loops, arithmetic and string parsing skills