1. Attendance
2. NVF
3. Go over syllabus and college policies
4. How to contact me
5. Discuss the book. How many people will not be using the book.
6. Objectives for tonight
   1. use strings for terminal input/output of text
   2. use int and float in arithemetic operations
   3. construct arithmetic expressions
   4. initialize and use variables with appropriate names
   5. discuss python coding standards
   6. import functions from library modules
   7. call functions with arguments; use return values appropriately
   8. construct several simple Python programs that perform input, calculations and output
   9. use docstrings to document python programs

The Software development Process Overview

1. Customer request – programmer receives broad statement of problem to be solved. Also known as requirements phase. Different options are evaluated and a plan is made.
2. Analysis – determine what the program will do and clarify the request. Try to answer as many questions up front before coding begins.
3. Design – determine how the program will perform it’s tasks
4. Implementation/Coding – develop the solution
5. Testing – make sure it works. Consider all possible scenarios
6. Integration – put it together with other program it may have to work with
7. Maintenance – handle changes to requirements, make modifications as needed, fix issues that got past testing
   1. Story: IRS still relies on WinXP and 20 year old fraud sniffing software
   2. <http://hothardware.com/news/recently-hacked-irs-still-relies-on-windows-xp-and-19-year-old-fraud-sniffing-software>
   3. Banks rely on old software
8. The waterfall model of the software development process (draw chart from p. 41)
   1. still used but outdated model where one phase is completed before the next one starts
   2. Agile is more dynamic and has started to replace waterfall
9. Case study: Income tax calculator:
   1. Goal: Illustrate the software development process for a small application and to create that application in Python
   2. Request
      1. customer requests program to compute person’s income tax
   3. Analysis
      1. Learn about the problem domain
      2. see p. 44 for tax laws used in the program
   4. Design
      1. Write an algorithm to solve the problem
      2. translate the algorithm to pseudocode (p. 44)
   5. Implementation (Coding) Phase
      1. translate the pseudocode to python
      2. Create the program that is on p. 45 – 46
      3. Explain that pseudocode and other design elements make it easier for a programmer to create a program that carries out the required computation. Pseudocode also allows for greater coding accuracy.
      4. Stress that it is always important to provide descriptive comments to document a program.
   6. Testing
      1. a single run is not an adequate test. We need to try various inputs and verify the outputs
      2. Testing is a deliberate process
      3. a good tester is a programmers best friend. He ensures the customer will not find any errors.
      4. A correct program produces expected output from any legitimate input.
      5. The goal of testing is to catch logic errors
      6. Use the test data on p. 47

**Getting Started with Python Programming**

1. Development Environment
   1. <http://docs.python-guide.org/en/latest/dev/env/>
   2. IDLE
   3. Eclipse PyDev
   4. PyCharm
   5. Notepad++
2. What is Python?
   1. Python is an interpreted, interactive, object-oriented programming language. It incorporates modules, exceptions, dynamic typing, very high level dynamic data types, and classes. Python combines remarkable power with very clear syntax. It has interfaces to many system calls and libraries, as well as to various window systems, and is extensible in C or C++. It is also usable as an extension language for applications that need a programmable interface. Finally, Python is portable: it runs on many Unix variants, on the Mac, and on PCs under MS-DOS, Windows, Windows NT, and OS/2.
3. Note that Guido van Rossum invented the Python programming language in the early 1990s.
4. Here's a very brief summary of what started it all, written by Guido van Rossum:

I had extensive experience with implementing an interpreted language in the ABC group at CWI, and from working with this group I had learned a lot about language design. This is the origin of many Python features, including the use of indentation for statement grouping and the inclusion of very-high-level data types (although the details are all different in Python).

I had a number of gripes about the ABC language, but also liked many of its features. It was impossible to extend the ABC language (or its implementation) to remedy my complaints -- in fact its lack of extensibility was one of its biggest problems. I had some experience with using Modula-2+ and talked with the designers of Modula-3 and read the Modula-3 report. Modula-3 is the origin of the syntax and semantics used for exceptions, and some other Python features.

I was working in the Amoeba distributed operating system group at CWI. We needed a better way to do system administration than by writing either C programs or Bourne shell scripts, since Amoeba had its own system call interface which wasn't easily accessible from the Bourne shell. My experience with error handling in Amoeba made me acutely aware of the importance of exceptions as a programming language feature.

It occurred to me that a scripting language with a syntax like ABC but with access to the Amoeba system calls would fill the need. I realized that it would be foolish to write an Amoeba-specific language, so I decided that I needed a language that was generally extensible.

During the 1989 Christmas holidays, I had a lot of time on my hand, so I decided to give it a try. During the next year, while still mostly working on it in my own time, Python was used in the Amoeba project with increasing success, and the feedback from colleagues made me add many early improvements.

In February 1991, after just over a year of development, I decided to post to USENET. The rest is in the Misc/HISTORY file.

1. Why is it called Python?
   1. At the same time he began implementing Python, Guido van Rossum was also reading the published scripts from "Monty Python's Flying Circus" (a BBC comedy series from the seventies, in the unlikely case you didn't know). It occurred to him that he needed a name that was short, unique, and slightly mysterious, so he decided to call the language Python.
2. Explain that Python is a high-level, general-purpose programming language for solving problems on modern computer systems.
   1. Python is a high-level general-purpose programming language that can be applied to many different classes of problems.
   2. The language comes with a large standard library that covers areas such as string processing (regular expressions, Unicode, calculating differences between files), Internet protocols (HTTP, FTP, SMTP, XML-RPC, POP, IMAP, CGI programming), software engineering (unit testing, logging, profiling, parsing Python code), and operating system interfaces (system calls, filesystems, TCP/IP sockets). Look at the table of contents for [The Python Standard Library](https://docs.python.org/2/library/index.html#library-index) to get an idea of what’s available. A wide variety of third-party extensions are also available.
   3. Python is an interpreted language.
      1. Programs are actually compiled to byte-code before being executed, but the byte-code is then interpreted. In some ways this is similar to Java or .NET which also compile to byte-code. Because these languages are statically typed with JIT compilers, they are generally considered as compiled languages whilst Python is generally considered as being interpreted.
   4. Python Success Stories
      1. <https://www.python.org/about/success/>
   5. Why learn python?

Python also acknowledges that speed of development is important. Readable and terse code is part of this, and so is access to powerful constructs that avoid tedious repetition of code. Maintainability also ties into this - LoC may be an all but useless metric, but it does say something about how much code you have to scan, read and/or understand to troubleshoot problems or tweak behaviors.

Python excels at toolmaking

" sufficiently powerful vs ease of use

" rapid development

" ease with which a programmer of other languages can pick up basic Python skills

" maintainability and easy to read code (unlike perl)

" huge standard library

" ease of setup of development environment

" interactive environment like iPython and IDLE

" it's free and has lots of free resources and has a large support community

" versatile programming language

" you're surrounded by data… and it keeps growing. Python is a great way to make sense of that data.

* 1. Why are you taking this class? What do you hope to learn?

**Running Code in the Interactive Shell**

1. Use Figure 1.6 and one or more real examples to show students how to write and run code in the Python Shell.
2. Demonstrate to students how they can get help through the command prompt or through the drop-down menu in the Python shell.
3. Explain and demonstrate how to quit the Python shell.

It's like this: Python leads a double life. It's an interpreter for scripts that you can run from the command line or run like applications, by double-clicking the scripts. But it's also an interactive shell that can evaluate arbitrary statements and expressions. This is extremely useful for debugging, quick hacking, and testing. I even know some people who use the Python interactive shell in lieu of a calculator!

Launch the Python interactive shell in whatever way works on your platform, and let's dive in with the steps shown here:

Example 1.5. First Steps in the Interactive Shell

>>> 1 + 1 1

2

>>> print 'hello world' 2

hello world

>>> x = 1 3

>>> y = 2

>>> x + y

3

The Python interactive shell can evaluate arbitrary Python expressions, including any basic arithmetic expression.

The interactive shell can execute arbitrary Python statements, including the print statement.

You can also assign values to variables, and the values will be remembered as long as the shell is open (but not any longer than that).

How to quit the inteactive shell:

File | Exit

**Interactive** **Python**

<http://interactivepython.org/runestone/static/pythonds/SortSearch/TheBubbleSort.html>

**Input, Processing, and Output**

1. Explain the concept of programming language syntax as it applies to Python.
2. Define the concept of a variable, and explain how one can be created to store user input.
3. Define the concept of type conversion functions, and explain how they can be used to make the user input suitable for the program's needs.
4. Use a few examples to show students how to perform simple input and output operations in Python.

**Editing, Saving, and Running a Script**

1. Explain the concept of a script as it applies to Python programs.
2. Use Figures 1.7 and 1.8 to explain how to edit, save, and run scripts in an IDLE window.
3. Define the concept of a program library, and explain how such libraries can be used.

**Behind the Scenes: How Python Works**

1. **Use Figure 1.9** to describe how Python code is interpreted and executed in the *Python Virtual Machine* (*PVM*).

Python code is interpreted and executed in the Python Virtual Machine (PVM)

What other programming languages work like this?

perl, java, c#

Which languages are different and compile to machine code?

1. c, c++

|  |  |
| --- | --- |
| *Teaching*  ***Tip*** | A good Python programming tutorial is available at <http://docs.python.org/tut/>. |

**Detecting and Correcting Syntax Errors**

1. Explain what the *syntax* of a programming language is.
2. Use one or more examples to stress that when Python encounters a syntax error in a program, it halts execution with an error message.

**Strings, Assignment, and Comments**

1. Use this section to introduce the use of strings for the output of text and the documentation of Python programs.

**Data Types**

http://www.voidspace.org.uk/python/articles/python\_datatypes.shtml#other-useful-values

1. Introduce the terms *data type*, *literal*, and *numeric data type*, emphasizing the differences and relationship between them. Table 2.2 lists some Python data types.

**statically typed language**

A language in which *types are fixed at compile time*. Most statically typed languages enforce this by requiring you to declare all variables with their datatypes before using them. **Java and C** are statically typed languages.

**dynamically typed language**

A language in which *types are discovered at execution time*; the opposite of statically typed. **VBScript and Python are dynamically typed**, because they figure out what type a variable is when you first assign it a value.

**strongly typed language**

A language in which **types are always enforced**. *Java and Python are strongly typed*. If you have an integer, you can't treat it like a string without explicitly converting it.

**weakly typed language**

A language in which *types may be ignored*; the opposite of strongly typed. VBScript is weakly typed. In VBScript, you can concatenate the string '12' and the integer 3 to get the string '123', then treat that as the integer 123, all without any explicit conversion.

So Python is both dynamically typed (because it doesn't use explicit datatype declarations) and strongly typed (because once a variable has a datatype, it actually matters).

**Python Data Types**

Python is a dynamically typed language. In a dynamically typed language a variable can refer to any kind of object at any time. When the variable is used, the interpreter figures out what kind of object it is. Java is a statically typed language. In a statically typed language the association between a variable and the type of object the variable can refer to is determined when the variable is declared. Once the declaration is made it is an error for a variable to refer to an object of any other type.

Python has five standard data types

* Numbers
  + int (signed integers)
  + long (long integers, they can also be represented in octal and hexadecimal)
  + float (floating point real values)
  + complex (complex numbers)
* String
* List
* Tuple (immutable list)
* Dictionary (key-value list like a hashtable)

**String Literals**

1. Explain the term string literal and use real examples to illustrate how to create string literals in Python.
2. Discuss None, True and False
   1. <http://python-history.blogspot.com/>
3. Introduce the term *empty string*, and be sure to distinguish the empty string from a string containing only white spaces.

|  |  |
| --- | --- |
| ***Teaching***  ***Tip*** | For more information about string literals, visit <http://docs.python.org/ref/strings.html>. |

1. Show students how to create strings that span multiple-lines, and introduce them to the *newline character*, \n.

**Escape Sequences**

1. Define the term *escape sequence*, and use Table 2.3 to introduce some of the most useful escape sequences in Python.

**String Concatenation**

1. Using one or more examples, demonstrate how to perform string concatenation in Python using the + operator.
2. Explain that the \* operator allows you to build a string by repeating another string a given number of times.

**Variables and the Assignment Statement**

1. Remind students the meaning of the term *variable*.
2. Explain the naming rules that apply to variables in Python. Provide several examples to illustrate each of these rules.

|  |  |
| --- | --- |
| ***Teaching***  ***Tip*** | CamelCase is a standard identifier naming convention for several programming languages. For more information, visit <http://www.c2.com/cgi/wiki?CamelCase>. |

1. Introduce the term *symbolic constant*, and note that programmers usually use all uppercase letters to name symbolic constants.
2. Use one or more examples to show how to write an *assignment statement* in Python. Introduce the term *variable reference* and explain the difference between *defining* or *initializing* a variable and variable references.
3. Explain the purposes variables have in a program. Be sure to explain the term *abstraction* in this context.

**Program Comments and Docstrings**

1. Explain the purpose and importance of *program comments*.
2. Provide examples of how to include *docstrings* and *end-of-line* comments in Python. Stress that using these types of comments appropriately is important.
3. Review the guidelines for creating good documentation of code. You can use the list on Page 53 of the text as a guide.

|  |  |
| --- | --- |
| ***Teaching***  ***Tip*** | The pydoc module can be used to display information about a Python object, including its docstring. For more information, visit <http://epydoc.sourceforge.net/docstrings.html>. |

**Quick Quiz 1**

1. Modern software development is usually incremental and \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

Answer: iterative

1. What is a correct program?

Answer: A correct program produces the expected output for any legitimate input.

1. True or False: In programming, a data type consists of a set of values and a set of operations that can be performed on those values.

Answer: True

1. The newline character \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ is called an escape sequence.

Answer: \n

**Numeric Data Types and Character Sets**

1. Explain that the following section provides a brief overview of numeric data types and their cousins, character sets.

**Integers**

1. Explain how integer literals are written in Python.

**Floating-Point Numbers**

1. Explain that real numbers have infinite precision, and point out that this is not possible on a computer due to memory limits.
2. Use Table 2.4 to introduce the use of *floating-point* numbers in Python. Explain that these numbers can be written using *decimal notation* or *scientific notation*.
3. Why are floating point numbers so inaccurate?
   1. http://docs.activestate.com/activepython/2.6/faq/general/index.html#python-s-design

|  |  |
| --- | --- |
| ***Teaching***  ***Tip*** | Complex numbers are also supported in Python. For more information, visit <http://docs.python.org/library/stdtypes.html>. |

**Character Sets**

1. Explain that in Python, character literals look just like string literals and are of the string type. Point out that character literals belong to several different *character sets*, among them the *ASCII set* and the *Unicode set*.
2. Provide a brief overview of the ASCII character set, using Table 2.5 as a guide.

|  |  |
| --- | --- |
| ***Teaching***  ***Tip*** | You can find more information about Python’s Unicode support at <http://www.amk.ca/python/howto/unicode>. |

1. Use a few examples to show how to convert characters to and from ASCII using the ord and chr functions.

**Expressions**

1. Explain that *expressions* provide an easy way to perform operations on data values to produce other values.
2. Note that when entered at the Python shell prompt, an expression’s operands are evaluated first. The operator is then applied to these values to compute the final value of the expression.

**Arithmetic Expressions**

1. Introduce the term *arithmetic expression*. Use Table 2.6 to describe the arithmetic operators available in Python.
2. Explain the difference between binary operators and unary operators. Give examples of each type of operator.
3. Briefly list the *precedence rules* that apply to arithmetic operators, using the list on Pages 58-59 as a guide. Be sure to explain the meaning of the terms *left associative* and *right associative*, and to point out that you can use parentheses to change the order of evaluation in an arithmetic expression.
4. Use Table 2.7 to show a few examples of how arithmetic expressions are evaluated. Introduce the term *semantic error* and clearly explain the difference between a semantic error and a syntax error.
5. Stress that when both operands of an expression are of the same numeric type, the resulting value is of that type; when each operand is of a different type, the resulting value is of the more general type.
6. Explain how the backslash character \ can be used to break an expression onto multiple lines.

**Mixed-Mode Arithmetic and Type Conversions**

1. Use a few examples to show how *mixed-mode arithmetic* can be problematic. Show how to use a *type conversion function* (see Table 2.8) to solve this problem.
2. Stress that the int function converts a float to an int by truncation, not by rounding. Show how to use the round function in cases when rounding is desirable.
3. Use one or more examples to show that type conversion also occurs in the construction of strings from numbers and other strings. Explain how to use the str function to solve this problem.
4. Explain that Python is a *strongly typed programming language*.

**Quick Quiz 2**

1. Real numbers have \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ precision, which means that the digits in the fractional part can continue forever.

Answer: infinite

1. True or False: In the 1960s, the original ASCII set encoded each keyboard character and several control characters using the integers from 0 through 255.

Answer: False

1. True or False: The precedence rules you learned in algebra apply during the evaluation of arithmetic expressions in Python.

Answer: True

1. Exponentiation and assignment operations are right \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, so consecutive instances of these are evaluated from right to left.

Answer: associative

**Using Functions and Modules**

1. Explain that Python includes many useful functions, which are organized in libraries of code called *modules*.

|  |  |
| --- | --- |
| ***Teaching***  ***Tip*** | Python’s global module index is available at <http://docs.python.org/modindex.html>. |

|  |  |
| --- | --- |
| ***Teaching***  ***Tip*** | “A module can contain executable statements as well as function definitions. These statements are intended to initialize the module. They are executed only the first time the module is imported somewhere.” *Reference:* <http://docs.python.org/tut/node8.html> |

**Calling Functions: Arguments and Return Values**

1. Introduce the terms *function*, *argument/parameter* (*optional* and *required*), and *returning a value*.
2. Define the concept of a *default behavior* of a function, and explain how this may be changed by calling the function using optional arguments.
3. Use an example to show how to obtain more information about a function by using help.

**The math Module**

1. Show how to use functions in the math module, both by importing the whole module and by importing individual resources.

**The Main Module**

1. Explain that, like any module, the *main module* can be imported. Show how this is equivalent to importing a Python script as a module.
2. Use the example provided in the book to show how to import the main module created in the case study of this chapter.

**Program Format and Structure**

1. Provide some guidance on how a typical Python program should look. Use the bullet points on Pages 67-68 as a guide.

**Running a Script from a Terminal Command Prompt**

1. Use Figures 2.6 and 2.7 to show how to run a script from a terminal command prompt.
2. Note that Python installations enable you to launch Python scripts by double-clicking the files from the OS’s file browser. Explain what the fly-by-window problem is, and how to solve it: add an input statement at the end of the script that pauses until the user presses the ENTER or RETURN key.

**Quick Quiz 3**

1. What is a module?

Answer: Python includes many useful functions, which are organized in libraries of code called modules.

1. A(n) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ is a chunk of code that can be called by name to perform a task.

Answer: function

1. True or False: Arguments are also known as literals.

Answer: False

1. The statement \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ would import all of the math module’s resources.

Answer: from math import \*

**Class Discussion Topics**

1. Are your students familiar with other software development models? If so, ask them to discuss their similarities and differences with the waterfall model. Which one do they like best?
2. Have your students used functions in other languages?

**Additional Projects**

1. Ask students to do some research and find out what other escape sequences are available in Python. They should compile a list of each of these with a short description of their functions.
2. Python provides other useful modules besides the math module. Ask students to do some research and find one or two other modules that they think will be useful in future projects. They should provide a brief description of each module, and include a list of two to five functions that belong to each module.

**Additional Resources**

1. String Literals:

<http://docs.python.org/ref/strings.html>

1. Python Docstrings:

<http://epydoc.sourceforge.net/docstrings.html>

1. Unicode HOWTO:

<http://www.amk.ca/python/howto/unicode>

1. Global Module Index:

<http://docs.python.org/modindex.html>

1. Module tutorial:

<http://docs.python.org/tutorial/modules.html>

**Key Terms**

* **abstraction:** A simplified view of a task or data structure that ignores complex detail.
* **analysis:** The phase of the software life cycle in which the programmer describes what the program will do.
* **Argument(s):** A value or expression passed in a method call.
* **arithmetic expression:** A sequence of operands and operators that computes a value.
* **ASCII character set:** The American Standard Code for Information Interchange ordering for a character set.
* **assignment operator:** The symbol =, which is used to give a value to a variable.
* **assignment statement:** A method of giving values to variables.
* **character set:** The list of characters available for data and program statements.
* **coding:** The process of writing executable statements that are part of a program to solve a problem. See also implementation.
* **comments:** Nonexecutable statements used to make a program more readable.
* **concatenation:** An operation in which the contents of one data structure are placed after the contents of another data structure.
* **correct program:** A program that produces an expected output for any legitimate input.
* **data type(s):** A set of values and operations on those values.
* **default behavior:** Behavior that is expected and provided under normal circumstances.
* **design:** The phase of the software life cycle in which the programmer describes how the program will accomplish its tasks.
* **design error:** An error such that a program runs, but unexpected results are produced. Also referred to as a logic error.
* **docstring:** A sequence of characters enclosed in triple quotation marks (“““) that Python uses to document program components such as modules, classes, methods, and functions.
* **empty string:** A string that contains no characters.
* **end-of-line comment:** Part of a single line of text in a program that is not executed, but serves as documentation for readers.
* **escape sequence:** A sequence of two characters in a string, the first of which is /. The sequence stands for another character, such as the tab or newline.
* **expression:** A description of a computation that produces a value.
* **float:** A Python data type used to represent numbers with a decimal point, for example, a real number or a floating-point number.
* **floating-point number:** A data type that represents real numbers in a computer program.
* **function(s):** A chunk of code that can be treated as a unit and called to perform a task.
* **Implementation:** The phase of the software life cycle in which the program is coded in a programming language.
* **integer:** A positive or negative whole number, or the number 0. The magnitude of an integer is limited by a computer’s memory.
* **integer arithmetic operations:** Operations allowed on data of type int. These include the operations of addition, subtraction, multiplication, division, and modulus to produce integer answers.
* **left associative:** The property of an operator such that repeated applications of it are evaluated from left to right (first to last).
* **literal:** An element of a language that evaluates to itself, such as 34 or “hi there.”.
* **logic error:** See design error
* **main module:** The software component that contains the point of entry or start-up code of a program.
* **mixed-mode arithmetic:** Expressions containing data of different types; the values of these expressions will be of either type, depending on the rules for evaluating them.
* **module(s):** An independent program component that can contain variables, functions, and classes.
* **newline character:** A special character (‘\n’) used to indicate the end of a line of characters in a string or a file stream.
* **optional arguments:** Arguments to a function or method that may be omitted.
* **Parameter(s):** See argument(s)
* **precedence rules:** Rules that govern the order in which operators are applied in expressions.
* **prototype:** A trimmed-down version of a class or software system that still functions and allows the programmer to study its essential features.
* **pseudocode:** A stylized half-English, half-code language written in English but suggesting program code.
* **required arguments:** Arguments that must be supplied by the programmer when a function or method is called.
* **returning a value:** The process whereby a function or method makes the value that it computes available to the rest of the program.
* **scientific notation:** The representation of a floating-point number that uses a decimal point and an exponent to express its value.
* **semantic error:** A type of error that occurs when the computer cannot carry out the instruction specified.
* **semantics:** The rules for interpreting the meaning of a program in a language.
* **software development life cycle (SDLC):** The process of development, maintenance, and demise of a software system. Phases include analysis, design, coding, testing/verification, maintenance, and obsolescence.
* **string(s) (string literals):** One or more characters, enclosed in double quotation marks, used as a constant in a program.
* **strongly-typed programming language:** A language in which the types of operands are checked prior to applying an operator to them, and which disallows such applications, either at run time or at compile time, when operands are not of the appropriate type.
* **symbolic constant:** A name that receives a value at program start-up and whose value cannot be changed.
* **test suite:** A set of test cases that exercise the capabilities of a software component.
* **type conversion function:** A function that takes one type of data as an argument and returns the same data represented in another type.
* **Unicode:** A character set that uses 16 bits to represent over 65,000 possible characters. These include the ASCII character set as well as symbols and ideograms in many international languages.
* **variable:** A memory location, referenced by an identifier, whose value can be changed during execution of a program.
* **variable reference:** The process whereby the computer looks up and returns the value of a variable.
* **waterfall model:** A series of steps in which a software system trickles down from analysis to design to implementation. See also software development life cycle.

## Session 2 October 8, 2015

Review:

## What is Python?

A programming language and a tool that will help you automate a number of common tasks on your computer. Python allows you to set up a series of related instructions which will produce the expected result.

Python is a scripting programming language known for both its simplicity and wide breadth of applications. For this reason it is considered one of the best languages for beginners. Used for everything from Web Development to Scientific Computing (and SO much more), Python is referred to as a “general purpose” language by the greater programming community.

## How did you hear about Python?

Do you have friends/colleagues using Python? What are they doing with it?

## What is a Python program?

a series of one or more text files each of which contains computer instructions

the instructions are processed by the Python interpreter. Windows knows to invoke the python interpreter because the file extension is .py.

each program consists of input | processing | output and is made up of statements that are either sequence or repetition (loops) or flow of control (decision) statements

When our program works with information we store it in variables. Variables can hold any data type but once you put some data in a variable then Python treats it as the type of data it is.

## Review: What do we know about Python so far?

The first step in writing a program is planning it

Programs are written in text documents using anything like notepad or an IDE

Spacing is important

Strings go in quotes

You can put special characters inside strings if you escape them (p. 50)

variables must begin with letter or \_ and cannot contain spaces

variables that don’t change use all upper case. These are called constants.

Comments begin with a # or can be between ‘’’ or “””

numbers without decimals are integers, with decimals are called floating point numbers

you can convert a number to a string with str() which allows you to append the number to the string

you can convert strings or numbers to int() or float()

you can round a number with round(6.5)

You can get help on a function with help(round)

you can import other modules with import math

functions take some input and return some output. Python will allows to create functions to modularize our code.

## Let’s try to change the Python version for PyCharm

Review the programs we created last class

How many minutes are in a year?

use range to return a sequence of numbers from start to stop by step

the last number in range is not returned. So range(1,5) goes from 1,2,3,4

range(stop) -> range object

| range(start, stop[, step]) -> range object

|

| Return a sequence of numbers from start to stop by step.

for eachValue in (range(2,20,2)):

print(eachValue)

## Let’s solve some problems together with Python

'''

How tall are four-story tall waves?

<http://www.washingtonpost.com/business/economy/cargo-ship-likely-sank-during-hurricane-coast-guard-says/2015/10/05/2bab59ae-6b79-11e5-aa5b-f78a98956699_story.html>

Python can help us find out how tall a four story building is

and how many of you stacked up would be the height of The Burj Khalifa in Dubui, UAE

<https://en.wikipedia.org/wiki/List_of_tallest_buildings_and_structures_in_the_world>. Multiply the number of people \* the average weight for a person. Would you rather be the guy on the top of the pile or at the bottom?

'''

#<http://www.ctbuh.org/TallBuildings/HeightStatistics/HeightCalculator/OnLineCalculator/tabid/1068/language/en-GB/Default.aspx>

FEET\_PER\_METER = 3.28084

nmbr\_stories = int( input("Enter the number of stories: "))

height = (3.9 \* nmbr\_stories) + 11.7 + (3.9 \* (nmbr\_stories/20))

#convert meters to feet

height = height \* FEET\_PER\_METER

#<https://docs.python.org/2/library/string.html#format-string-syntax>

print("The {0} story tall building would be {1:,.0f} feet tall".format(nmbr\_stories,height))

'''

You may be as astonished as we were to learn how much water runs off a roof in a rain storm!

Take a guess--what is the volume of water that runs off a 1,000 square foot roof during a 1" rain storm?

20 gallons? 50 gallons? 100 gallons? 1000 gallons?

Python can give us the answer.

To calculate the runoff from any given rainfall:

Take the dimensions of the footprint of your roof and convert them to inches. (So, a 50' x 20' roof is 600" x 240".)

Multiply the roof dimensions by the number of inches of rainfall.

In this example, 600" x 240" x 1" = 144,000 cubic inches of water.

Divide by 231 to get the number of gallons (because 1 gallon = 231 cubic inches). (144,000/231 = 623.38).

'''

length = float(input("Enter the length of your roof (feet): "))

width = float(input("Enter the width of your roof (feet): "))

area = (length \* 12) \* (width \* 12) #calculate the area of the roof in sq. inches

how\_much\_rain = float(input("How much rain fell? "))

cu\_inches\_water = area \* how\_much\_rain

nmbr\_gallons = cu\_inches\_water/231 #because 1 gallon is 132 cu inches

print("Approximately {0:,.1f} gallons of water fell on your roof when {1:.1f} inches rain fell.".format(nmbr\_gallons,how\_much\_rain))