# Course outline for Python training

Programming in Python
Dave Kuhlman

# Intro, admin, etc

- Introductions
- Administration and other practical matters
- Course agenda, contents, purpose, and goals
- Machine set-up and materials

# Python (and other languages)

- Python
- Comparisons with other languages
- Connections to other languages
- Versions of Python
- Why Python?  Also, when and why not Python.

# Course Preview

- Lexical matters
- Built-in data types – numbers, strings, lists, dictionaries, files, etc.
- Statements – assignment, if:, for:, while:, etc
- Functions
- Classes and objects
- Other topics: list comprehensions, iterators, standard modules, etc
- And, ”practical” exercises everywhere

# Lexical Matters

- Names
- Scope
- Keywords and operators
- Line structure
- Comments
- Statement structure
- Blocks and indentation
- Special names

# Built-in datatypes

- Numbers
- Strings
- Lists and tuples
- Dictionaries
- Files
- None, booleans, sets, functions, methods, classes, modules, type objects, etc.

# Objects and data types

- Mutability and immutability
- Objects have attributes.  Some attributes are methods.
- Use *type(obj)* and *dir(obj)*
- References and sharing.  Use *id()* and the *is* operator to check identity.

# Numbers

- Integers
- Longs (really long)
- Floats
- Complex/imaginary
- Literal representations
- Operators
- Mixed arithmetic

# Strings

- What are strings? A string is an immutable ordered sequence of characters.
- String literals – single quotes, double quotes, and triple quoting
- String operators
- String methods
- String Formatting

# Lists and tuples

- Lists and tuples are sequences – ordered collections.
- Lists vs. tuples – mutable and immutable
- Literal representation of lists and tuples
- Operators
- Methods

# Dictionaries

- What are dictionaries? – An unordered collection of key-value pairs.  A mapping from keys to values.
- Literal representation of dictionaries
- Operators for dictionaries
- Methods for dictionaries

# Container summary

- Strings – ordered, characters, immutable
- Tuples – ordered, heterogeneous, immutable
- Lists – ordered, heterogeneous, mutable
- Dictionary – unordered, key/values,  mutable
- Set – unordered, heterogeneous, mutable, unique values

# Files

- What is a file object?  A file object represents a file on the file system.  There are other file-like objects.
- Creating a file object – open() – name/path, mode (read, write, append)
- Using a file object – File methods: read, readlines, write, seek, etc.
- Iterating over the lines in a text file.

# A few other data types

- The null type – None
- Boolean values – True and False
- Sets
- Functions and methods
- Classes and instances of classes
- Modules
- Additional builtin and defined types in the Python standard library

# Introduction to Statements

- Statements are executed.
- Statements do not return a value.
- Compound statements have a header and contain one or more blocks.

# Statements

- Simple statements -- assignment, print, import, break, continue, raise, ...
- Compound statements – if:, for:, while:, try:, with:, ...

# Functions: Introduction

- Introduction to functions
- Why functions – task abstraction; application organization/structure.
- Execution model for functions
- Functions are first-class objects

# Function definition

- Function block header
- Parameters – plain
- Parameters – default values
- Parameters – arg lists and keyword arguments (*args, **kwargs)
- Returning values from a function
- Scope – Bindings within a function.  Also, the *global* statement.

# Using/calling functions

- Passing arguments – plain
- Passing arguments – keyword
- Using function return values
- Functions by name, functions by variable, functions in data structures
- Unpacking return values
- Unrolling collections in a function call

# Object-oriented programming

- Why OOP? -- object abstraction; modeling objects in the real world; reuse.
- Organization – classes are another way to structure and organize your code.
- Encapsulation
- Data(and method) hiding
- Inheritance
- Polymorphism – duck typing

# Defining classes

- Class block header: name and superclass
- Adding methods
- The constructor: __*init*__
- Creating instance/member variables
- Class data members
- Class methods
- Properties

# Creating and using instances

- Creating an instance is like ”calling” a class -- parenthesis.
- Instances are mutable.
- Instances can be shared.
- Instances are first-class objects (classes, too).

# Instance (and class) variables

- What are instance variables?
- What are class variables?
- Initialization – Create instance variables in the constructor: __init__().

# Kinds of methods

- Instance methods
- Class methods
- Static methods

# Defining methods

- A method is a function in a class.
- ”self”
- Calling methods in this class or superclass
- Calling methods in a super-class and bypassing the same method in this class

# Inheritance and polymorphism

- Implementing inheritance
- How inheritance works – (1) Conceptual view; (2) mechanical view.
- Implementing and using inheritance.
- Polymorphism – (1) Other views of polymorphism;  (2) the ”Pythonic” view.
- Implementing and using polymorphism.
- Interfaces and ”duck typing” – Polymorphism not strictly tied to inheritance.

# Introducing modules

- Introduction – You likely already know each other.
- Why we need modules – structure.
- Modules enable us to group related programming objects: functions, classes, variables/names.
- Modules are the next highest structural artifact above functions and classes.
- Import creates module objects, only once.

# Defining modules

- A module is a .py file.
- Dual use – Both import and run a module.
- What to put in a module: variables, functions, classes, run/test harness, other statements.
- Doc-string – Triple quoted string at top of module.

# Using modules

- *import* – Evaluates a .py file and creates a module object containing other objects: names bound to objects, function definitions, class definitions, etc.
- Reference objects in a module using standard dot notation.  Or, use *from x import y.*
- When imported, a module executes only once. Module objects are shared.