Lecture 2 - Objects and Types

Wednesday, August 30, 2017 8:59 AM

- Objects are spaces in memory. They store data
  - Has a type
    - Strings (str), Integers (int, long), Floats (float), Boolean (bool)
    - Determine all valid operations
  - Also has a value, the information of interest

Lecture 3 - Expressions

Friday, September 1, 2017 9:06 AM

- **Variables**
  - Used to give values a name
  - Choose meaningful names
    - All lowercase letters by convention.
    - Must start with letter, but can contain numbers and underscores
    - Underscores used in lieu of spaces
    - Cannot be keywords
- Combine operators and values to form **Expressions**
  - Simple syntax:
    - <value> <operator> <value>
  - Results in a **new value**
  - Numbers support six basic operators
    - Add, subtract, multiply, divide, exponentiation, and modulo
  - Order of operations - PEMDAS
    - **Not** left to right, UNLESS multiple operations of same precedence
      - 4 / 2 / 2 = 1
  - Only use () to group values. [] and {} mean specific, different things
  - Comparison operators return Booleans and support numbers **and** strings
  - Useful string operators:
    - Concatenation using +
    - Repeating using *
    - Every string has an internal structure.
    - Indexing a String
      - 'msg = "Python is awesome"'
      - 'print msg[0]' P
      - 'print msg[0:7]' Python
      - Left index is inclusive, right is exclusive

Lecture 4, 5, 6 - Functions

Wednesday, September 6, 2017 8:52 AM

- Functions are the main tool to package a set of related statements
- Used to avoid repeating statements which leads to bugs and low adaptability
- Many built in functions
  - Casting (str(*), float(*), int(*), etc)
  - type(*)
  - len(*)
  - raw_input(str)
  - round(float, int)
- **Libraries**
  - Modules which are collections of functions and variables
  - Must be imported.
  - Example: math has many functions and constants
- Defining functions
  - Specific format:
    - def name():
    - Statements
    - return
  - Can be defined as having an input, simply put a variable name in the () in def statement
    - These parameters are variables, not typed
    - Can have multiple, separated with commas
- To actually execute a function, you must **call** the function
  - Called by name, with parenthesis and any required parameters
- Functions can and should return values
  - Printing and raw_input should not take place inside a function
  - An empty return statement returns the special type "None"

Lecture 7 - Variable Scope

Wednesday, September 13, 2017 8:57 AM

- In Python, statements are executed one at a time, in order from top to bottom.
- Functions are not executed until called.
    - Functions are like a detour in the flow of execution.
        - The flow jumps to the body of the function, executes all statements there, then returns to where it left off
- The scope of a variable defines what part of the code can see/use the variable
- Functions are black boxes
    - The program outside the function can ONLY pass arguments and catch return values.
    - Whatever happens inside the function stays there
    - Variables created/assigned in the function are **local variables** which only exist inside the function
    - Arguments are values from outside world, parameters are variables created by function (essentially the same as saying param1 = arg1)
- A frame is essentially a table with two columns to keep track of variables.
    - Has a name column and a value column.
    - There is a main (global) frame for all code outside function definitions
    - Every time a function is called, a new empty frame is created
- Remember - assignments evaluate the right hand side and **then** make the assignment
- You can declare a local variable to be global. This is generally considered bad practice

Lecture 8 - Wrapping Up Functions

Friday, September 15, 2017 9:37 AM

- Composition of functions - Functions return values, so they can be used as the arguments of ANOTHER function call
    - Evaluate the innermost function call arguments first
- Functions can also be called from inside other functions. Must be defined first before called in another definition
- Return statements mark the end of the function. Any code following a return statement is dead code (unless it is conditional)

Lecture 9,10 - Conditionals

Monday, September 18, 2017 8:54 AM

- Basic structure:
  - If boolean_expression:
    - Statements
  - Elif other_expression:
    - Statements
  - Else:
    - Statements
- If statements DO NOT create a new frame. Same scope as container.

Lecture 11,12 - For Loops

Friday, September 22, 2017 8:59 AM

- Iteration
  - Repetition of statements
  - Each repetition is known as a pass or iteration
  - Definite or indefinite
    - Definite - for loops, set number of times
    - Indefinite - while loops, no idea how many times it will run, keeps going for condition
  - **For** loops
    - for i in [1,2,3]:
      - Statements
    - No need to write out explicit range:
      - Range function
  - If you are not using the loop variable, place an underscore

Lecture 13,14 - While Loops

Monday, September 25, 2017 9:00 AM

- While loops are indefinite. They continue running until a condition is False
- Syntax:
    - While boolean_expression:
        - Statements
    - Can loop infinitely if boolean_expression never becomes False

Lecture 15, 16, 17 - String Methods & Loop Examples

Monday, October 2, 2017 9:31 AM

- Slicing can include step size
  - str[::-1] will reverse a string
  - str[:] yields copy of the entire string
- You can directly iterate over strings (strings are a form of sequence, just like lists)
- Comparing booleans to the values True or False is almost always meaningless - use 'not' or just the variable itself
  - X = True
  - If X == True :: if x
  - If X == False :: if not x
- The operator 'in' can be used to check if a sequence is in another
- Methods:
  - Special functions for each type. Called on the variable itself, using a period
  - Str methods include lower(), find(str), replace(str,str), and more
  - Usually not 'called', methods are 'invoked'

Lecture 18 - Numeral Systems

Friday, October 13, 2017 9:01 AM

- Numbers are abstractions
  - 37 is only 37 because it was decided to be
    - 0x25 is the same value
- Digits
  - Depend on base. In base n, digits are 0 up to (and including) n-1
- Decimal System - Everyday usage to represent numbers
  - Base 10. 10^0, 10^1, 10^2
  - Digits:
    - 0 1 2 3 4 5 6 7 8 9
  - 137 is represented by 1, 3, and 7 being multiplied by various powers of 10
    - 1 * 10^2 + 3 * 10^1 +7 * 10^0 = 137
    - 137 = 13 * 10 + 7
      - 13 = 1 * 10 + 3
        - 1 = 0 * 10 + 1
      - Remainders, in reverse, are the number
        - This is not particularly illustrative, because it is base 10 -> base 10
- Binary System
  - Base 2
  - Digits:
    - 0 1
  - Dec: 13
    - 13 = 6 * 2 + 1
      - 6 = 3 * 2 + 0
        - 3 = 1 * 2 + 1
          - 1 = 0 * 2 + 1
    - Reading backwards, the binary representation of 13 is 1101
  - Binary 1101
    - 1 * 2^0 + 0 * 2^1 + 1 * 2^2 + 1 * 2^3 = 13
- Hexadecimal System
  - Base 16
  - Digits:
    - 0 1 2 3 4 5 6 7 8 9 A B C D E F
  - 2 digit hex numbers go up to 256
    - Take up eight bits (2^8)
      - This is a **byte**
  -
- Python:
  - bin() takes a number and returns a string
    - Starts with 0b, rest is binary representation of the input number
  - int() can take a string in any base. Second argument specifies base
  - hex() takes a number and returns a hex string
    - Starts with 0x
  - ord() takes a char and returns a number
    - 'ordinal numbers'
  - chr() takes a number and returns the character
- Representing characters as numbers
  - ASCII - Old style of representing characters using only 4 bits - 0 through 127, each number

corresponding to a character
- Capital numbers are associated with smaller numbers

Lecture 19, 20 - Lists

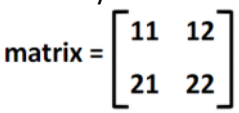
Monday, October 16, 2017 8:56 AM

- Data structure, collection of items
- Order matters
- Can have identical entries ([1,1]) is valid
- Tend to **like** having the same type in each list, **but** CAN be mixed in Python
  - We will not be mixing types
- Equivalent of an array in other languages (more or less)
- Lists can contain other lists
  - 'Nested' lists
- Lists are a SEQUENCE, like Strings
  - Can be indexed, sliced, used in len().
  - Can also use + (concatenation) and * (repetition)
- Lists are MUTABLE
  - Unlike strings, can be changed with indexing **and** slicing
  - List1 = [1,2,3]
  - List1[2] = 4
  - print List1
    - [1,2,4]
  - Setting a slice equal to a larger list will simply extend the list
- List methods:
  - Append: add element to the end of a list
  - Many of these are voids.
  - Remove: remove a **value**
  - Insert: insert a value at index insert(index,value)
  - Pop: remove a specific **index**
  - Index: returns the index of the specified value
  - Reverse: in-place reverse
  - Extend: pass a second list, will extend the first list by the second list
- The 'in' operator can be used in sequences
  - Value **in** list
- **Sequences can be iterated in for loops**
  - Can also iterate over indexes.
- Functions: sum, min, max
  - NOT methods
  - Return what they say when passed a list
- Strings and lists
  - List(str) splits a string into characters
  - Str.split(delimiter) makes a list of words which were separated by the delimiter (default " ")
  - Str delimiter.join(list). - joins items in a list seperated by the delimiter. ' '.join(list) undoes str.split()

Lecture 21 - Nested loops and lists

Friday, October 20, 2017 9:01 AM

- A nested loop is a loop inside another loop
- The outer loop increments once for every full run of the inside loop
- Mostly useful for **nested lists**
  - Nested lists are lists inside lists
  - Often used to represent matrices
  - Each entry in the list is itself a list representing the row of the matrix




```
matrix = [[11, 12], [21, 22]]
```

- Nested lists have length, which is still just the number of entries.
- They also have a **depth**, how many layers there are
- The above matrix has a length and depth of 2, but they are completely independent

Lecture 22 - Mutability

Monday, October 23, 2017 8:57 AM

- Mutability and Immutability
- Mutability
  - Mutable objects can be updated in place
  - Lists are mutable
  - Methods will often change the value and return None
    - For example, append adds an item to a list and returns None
  - Mutable objects in this course:
    - list, dict, set
- Immutability
  - Immutable objects **cannot** be updated in place once created
  - Strings are mutable
  - Need to create a new value and then bind the name to the new value
    - For example: all string methods return a **new** string
  - Immutable objects in this course:
    - int, long, float, str, tuple
- Aliasing
  - We can bind multiple names to the same object
  - This has important implications for mutability
    - Immutable objects cannot be edited in place, so aliasing them and then changing one name will not affect the other
      - This is because changing one rebinds it to a new value, so they no longer refer to the same thing
    - Mutable objects work differently. Two names pointing to the same list, then editing the list through either (IN PLACE), will update both
- Assignment is **not** an in-place change
  - Can't change aliased names
- Cloning
  - Use slicing notation [:] to get a copy that isn't the same object
- Loops and mutability
  - Should avoid removing elements of a list that is being iterated over in a list. Use a clone
- Functions and mutability
  - In place operations will affect the object **outside the function**
  - Operations that are not in place will have no affect
  - *Functions can have side effects on mutable objects*

Lecture 23 - Tuples

Wednesday, October 25, 2017 8:54 AM

- Very similar to lists, but **IMMUTABLE**
- Tuples are sequences. Len(), iteration in for loops, slicing, indexing, join, repeat, in, etc
- Denoted with () rather than []. For single element tuples, need trailing comma. ex: (1,)
    - () are optional
        - Tuple1 = 1,2 is just as valid as tuple1 = (1,2) - called tuple packing
- You can have multiple variables on the left side of an assignment, and a tuple on the other
    - Must have the same number of variables as values in the sequence
    - This is sequence unpacking
- Tuples are immutable, so tuple1[0] = 'Hello' will give an error
- Tuple methods:
    - Count(object) returns the number of times object occurs in the tuple
    - Index(object) returns the first place the object is found in the index
- Tuples and Lists:
    - Lists are conventionally all one data type, Tuples can be mixed
    - Tuples are typically short and are usually accessed by sequence unpacking, rather than lists which are often indexed in for loops
    - Lists are **mutable**, tuples are **not**
    - Go from a list to tuple and back: tuple() and list()
- A list of tuples can be interated over directly. For x,y,… in list1 will unpack the sequences (tuples) in list1 into x,y,…
- Zip():
    - Takes two or more sequences and zips them into a **list** of **tuples**
    - Zip(list1,list2) returns [(list1[0], list2[0]), (list1[1], list2[1]), …]
    - Cuts off at length of *shortest argument*
        - zip([1, 2, 3], [4, 5]) returns [(1, 4), (2, 5)]

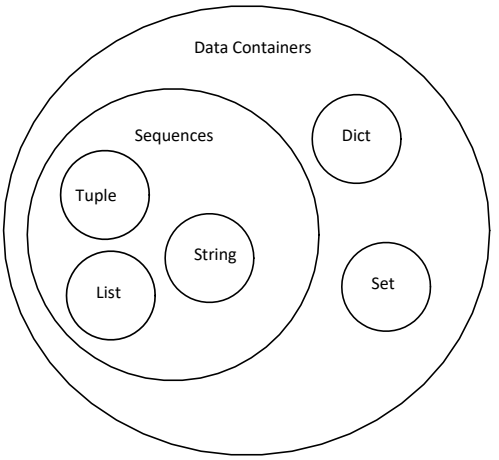
Lecture 24, 25 - Dictionaries and Sets

Friday, October 27, 2017 9:13 AM

- **Dictionaries**
  - Map between a set of unique keys and values (key-value pairs)
  - Keys must be **immutable** and **unique**
    - str, int, float, tuple (we only use int and str)
  - Values can be any type
  - Dicts are mutable
  - Defined by dict() or {}
  - [] are used to add or access values. Keys, NOT indexes
    - Use assignment to add key-value pairs
      - Dict1['Mark'] = 'red'
    - Key indexing accesses values
      - Dict1['Mark']
  - Orders of dictionaries are arbitrary
  - Attempting to access a key that is not defined yields errors
  - Can also use {}.get(key). Returns None if not present
  - Can use **in** operator, checks if **key** is inside the dictionary
  - Two methods, keys() and values(), return lists of their respective data
    - Lists are arbitrarily (but deterministically) sorted, however, both lists for keys() and values() are in the SAME order
    - The method items() returns a list of key-value tuples
- **Sets**
  - **Unordered** collection of **unique** (and immutable) elements
  - Sets are mutable
  - Set1 = { 'BC', 'NEU', 'BC' }
    - This will resolve to { 'BC', 'NEU'} - NO REPEATS
  - Empty sets are created with set(), NOT {}
  - Fast membership testing
  - Four methods
    - .add(object) - equivalent of append, adds a new element. In place
    - .update(other_set) - combine two sets. In place
    - .remove(object) - removes the object from the set. In place
    - .intersection(other_set) - **returns** a set with the common elements of the two sets

Sidebar:

- Strings, lists, and tuples were all sequences.
  - Order matters, can be indexed and sliced
- Sequences are part of a larger categorization, called data containers
  - Data containers can be used in for loops, can be used in len(), can use the in operator
- Dictionaries and sets are kinds of data container.



Lecture 26, 27, 28, 29 - Recursion

Wednesday, November 1, 2017 9:00 AM

- Recursion is a mathematical idea
  - f(n) = g(f(n-1))
- Breaking a problem into problems of the same type over smaller amounts
- Should have a base or trivial case and recursive case.
  - Consult code examples
- Frames - Each call creates a new frame
  - Recursion means many nested frames, which may mean one name has many values depending on which frame you are in
- It is a good idea to store results in a dictionary to speed up / prevent repeats. See fib_smarter in lecture29.py

Search Algorithms:

- Linear and binary. Binary is massively faster, but requires a sorted list
  - O(n) vs O(log(n))

Lecture 30, 31 - Search Algorithms

Friday, November 10, 2017 9:00 AM

Useful:
%timeit in Atom to analyze time to run

- **Selection Sort**
  - Select minimum values and move to front
    - Scan the list to find the smallest value, then swap this with the value at index 0
    - Scan the remaining values (starting at index 1), and find next smallest. Swap it with index 1
    - Continue this pattern until sorted
  - Time complexity - n loops to go through elements, n/2 on average to find min (gets shorter as you go along), so it is on average n^2 / 2, or O(n^2)
- **Insertion Sort**
  - Insert each value into the place it needs to be to be sorted
    - A list of one value is already sorted
    - Insert the second value where it should go to be sorted
    - Insert third into the list of two elements to be sorted
    - Continue until done
  - Time complexity - O(n^2) on average
- **Merge Sort**
  - Based on method to merge two sorted lists into a list that is also sorted
  - Key is breaking down list into smaller list, then merging them back up
  - Inherently recursive
    - A list of 1 element is sorted. Break each list in half until it's length 1, then return the merged halves all the way up
  - Time complexity - O(nlog(n)) - merge is linear, the algo is logarithmic like binary, so result is n* log(n)

Lecture 32 - Backtracking

Wednesday, November 15, 2017 8:59 AM

- Eight Queens Problem
  - How to arrange 8 queens on a chess board such that no queen can attack another
    - Strategy:
      - Place a queen in the first column
      - Place one in each column in a safe place until you're done or stuck
    - Backtracking:
      - If you get stuck, go back and remove the previous until a new path presents itself
    - This is usually done recursively, and immutable objects are useful for this
      - See the lecture 32 code
- Brute force - trying every single possible combination of solutions. Slow, but guaranteed to find solution
  - Slow and not clever. Fairly easy to implement
- Sudoku -
  - Brute force backtracking
  - Base case - no empty squares
  - When an empty cell is found, try 1-9 until you find a valid choice
  - If a legal number is found, recursively call the function on that potentially correct new board
  - Otherwise, backtrack (probably return none).

Lecture 33, 34 - File Processing (I/O)

Monday, November 20, 2017 9:03 AM

- Opening a file
  - Built in **open** function
    - Returns a file handle, an object used to perform operations on the file
      - Syntax is (filename,mode) mode is optional, either 'r' read, 'w' write, 'a' for append
      - Like a gate from memory to the hard drive
      - Use .close() method when done
      - Can think of a file object as a list of strings (IT IS NOT, but it acts similarly)
        - For line in file_object:
        - This includes hidden characters, including '\n' and other escape characters
          - Commonly use .rstrip() string method to remove
  - Writing to a file:
    - Opening in write mode will create a file if it doesn't exist
    - .write(str) - does not add \n, should do it manually
    - Writing to a file is the equivalent of deleting the file first then writing to it
      - Use append mode 'a'
- Working with folders
    - Use the os module
      - os.getcwd() - get current working directory
      - .path.isdir(str) and .path.isfile(str) return true if there is a directory/file of that name in CWD
        - Can give function an absolute path
      - .listdirs(cwd) - list directories in cwd
      - .path.exists(str) - returns true if the path exists

Lecture 35, 36 - Exception Handling

Wednesday, November 29, 2017 9:00 AM

- Error Categories
  - Syntax Error - Don't follow the language's rules
    - Code will not run
  - Runtime Errors - Occur when the program is running and Python encounters some code that cannot be executed.
  - Logic Errors - Occur when the program does not produce the expected result
- Runtime Errors
  - Cause Exceptions - expectations not followed
  - A exception is an event, which occurs during execution and disrupts the normal flow of the program
  - Common Causes:
    - Dividing by zero
    - Trying to read a nonexistant file
    - Indexing larger than the size of the sequence
  - Common Exceptions
    - Exception - superclass
    - ZeroDivisionError - division or modulo by 0
    - IOError - raised when an input/output operation fails
    - IndexError - raised when an index is not found
    - KeyError - key not found in a dictionary
    - UnboundLocalError - trying to access a local variable in the function, but not value has been assigned to that variable
    - TypeError - operation or function is applied to object of inappropriate type
  - Exception Handling
    - We can instruct Python on what to do when it encounters an exception
    - Use the **try** block

Lecture 37 - Higher Order Functions

Monday, December 4, 2017 8:55 AM

- Three higher order functions (concepts borrowed from functional programming)
    - Map, filter, reduce
- Functions are first-class objects in python. Can be treated like we treat other objects (strings, etc)
- A higher order function is a function that works on other functions
    - Generally takes a function as an argument

Lecture 38, 39 - List Comprehension

Wednesday, December 6, 2017 9:00 AM

- A Python construct which allows list creation by applying an expression to every item in a list
  - result = [some_expression **for** item **in** given_list]
  - Expression can be functions as well
  - Can add a condition
    - result = [some_expression **for** item **in** given_list **if** condition]
  - Can be nested, used in creative ways
    - Flatten a depth 2 nested list:
      - flat = [value for subject in nested for value in sublist]
    - Multiply elements in a matrix by 2
      - new_mtx = [[2*value for value in row] for value in mtx]
- In effect, comprehension can replace most **map** and **filter** behavior
- This also works for sets and dictionaries in its own way. Sets are an obvious extension