Computational Structures in Data Science

Lecture #11:
Iterators and Generators

Today:
• Review Exceptions
• Sequences vs Iterables
• Using iterators without generating all the data
• Generator concept
  — Generating an iterator from iteration with `yield`
• Magic methods
  — `next`
  — `iter`
• Iterators – the `iter` protocol
• `getitem` protocol
• Is an object iterable?
• Lazy evaluation with iterators

Summary of last week
• Approach creation of a class as a design problem
  — Meaningful behavior => methods [& attributes]
  — ADT methodology
  — What’s private and hidden? vs What’s public?
• Design for inheritance
  — Clean general case as foundation for specialized subclasses
• Use it to streamline development
• Anticipate exceptional cases and unforeseen problems
  — try ... catch
  — raise / assert

Key concepts to take forward
• Classes embody and allow enforcement of ADT methodology
• Class definition
• Class namespace
• Methods
• Instance attributes (fields)
• Class attributes
• Inheritance
• Superclass reference

Computational Concepts Toolbox
• Data type: values, literals, operations,
  - Expressions, Call expression
  - Variables
  - Assignment Statement, Tuple assignment
  - Sequences: tuple, list
  - Dictionaries
  - Function Definition Statement
  - Conditional Statement
  - Iteration: list comp, for, while
  - Lambda function expr.
• Higher Order Functions
  - Functions as Values
  - Functions with functions as argument
  - Assignment of function values
  - Higher order function patterns
  - Map, Filter, Reduce
  - Function factories – create and return functions
  - Recursion
  - Abstract Data Types
  - Mutation
  - Class & Inheritance
  - Exceptions
  - Iterators & Generators

Mind Refresher 1
An object is...
A) an instance of a class
B) a python thing
C) inherited from a class
D) All of the above

Solution:
A) An object is an instance of a class
Mind Refresher 2

A setter method...
A) constructs an object
B) changes the internal state of an object or class
C) is required by Python to access variables
D) All of the above

Solution:
B) Changes the internal state of an object or class by allowing access to a private variable.

Solution:
B) Changes the internal state of an object or class by allowing access to a private variable.

Exception (read 3.3)

• Mechanism in a programming language to declare and respond to “exceptional conditions”
  – enable non-local continuations of control
• Often used to handle error conditions
  – Unhandled exceptions will cause Python to halt and print a stack trace
  – You already saw a non-error exception — end of iterator
• Exceptions can be handled by the program instead
  – assert, try, except, raise statements
• Exceptions are objects!
  – They have classes with constructors

Handling Errors – try / except

• Wrap your code in try — except statements

```
try:
    <try suite>
except <exception class> as <name>:
    <except suite>
... # continue here if <try suite> succeeds w/o exception
```

• Execution rule
  – <try suite> is executed first
  – If during this an exception is raised and not handled otherwise
  – And if the exception inherits from <exception class>
  – Then <except suite> is executed with <name> bound to the exception

• Control jumps to the except suite of the most recent try that handles the exception

Types of exceptions

• TypeError — A function was passed the wrong number/type of argument
• NameError — A name wasn’t found
• KeyError — A key wasn’t found in a dictionary
• RuntimeError — Catch-all for troubles during interpretation
  ...
Mind Refresher 3

Exceptions...
A) allow to handle errors non-locally
B) are objects
C) cannot happen within a catch block
D) B, C
E) A, B

Solution:
B, C Exceptions are objects and they can occur any time.

Iterable - an object you can iterate over
- iterable: An object capable of yielding its members one at a time.
- iterator: An object representing a stream of data.
- We have worked with many iterables as if they were sequences

Functions that return iterables
- map
- range
- zip
- These objects are not sequences.
- If we want to see all of the elements at once, we need to explicitly call list() or tuple() on them

Generators: turning iteration into an iterable
- Generator functions use iteration (for loops, while loops) and the yield keyword
- Generator functions have no return statement, but they don’t return None
- They implicitly return a generator object
- Generator objects are just iterators

```python
def squares(n):
    for i in range(n):
        yield i*i
```

Next iteration
```
def all_pairs(x):
    for item1 in x:
        for item2 in x:
            yield (item1, item2)
```

Next element in generator iterable
- Iterables work because they have some "magic methods" on them. We saw magic methods when we learned about classes.
- e.g., __init__, __repr__ and __str__.
- The first one we see for iterables is __next__
- `iter()` – transforms a sequence into an iterator
Iterators – iter protocol

• In order to be iterable, a class must implement the iter protocol.
• The iterator objects themselves are required to support the following two methods, which together form the iterator protocol:
  – __iter__(): Return the iterator object itself. This is required to allow both containers and iterators to be used with the for and in statements.
  – __next__(): Return the next item from the container. If there are no further items, raise the StopIteration exception.
• Classes get to define how they are iterated over by defining these methods.

Getitem protocol

• Another way an object can behave like a sequence is indexing. Using square brackets "[]" to access specific items in an object.
• Defined by special method: __getitem__(self, i)
  – Method returns the item at a given index

Determining if an object is iterable

• from collections.abc import Iterable
• isinstance([1, 2, 3], Iterable)
  – This is more general than checking for any list of particular type, e.g., list, tuple, string...

Solutions for the Wandering Mind

N bits can represent $2^N$ configurations.
1) How many functions can be created that map from N bits to 1 bit (binary functions)? $\#functions=2^N$
2) How many functions can be created that map from N bits to M bits? $\#functions=M^{2^N}$
3) How many functions can be created that map from N k-bit length integers to M bits? $\#functions=M^{2^k}$
4) If we were representing the functions 1, 2, and 3 in tables: a) How many different tables would we need? $2^{2^N}$, $M^{2^N}$, $M^{2^k}$ tables, respectively.
b) How big is each table? $(N+1)\times2^N$ table cells.

=> It’s easier to abstract the tables using functions, abstract data types, and object orientation!

Questions for the Wandering Mind

1) Adding two n-bit integers, how many bits can the result have?
2) Multiplying two n bit integers, how many bits can the result have?

Assume:
a) Exceptions don’t exist
b) We only reserve 8-bit for an integer variable (0-255)

Questions:
1) What would be the result of an addition 255+255?
2) What would be the result of a multiplication 255*255?
3) Assume I additions of 8bit integers into the same 8bit variable. Can you formulate the maximum error that can occur as a function of I?