1MP3 Midterm 2 Review

14 November 2019

## reading files

* f = open(filename, "r"), f.close()
* f.read() reads the *entire file* as a string
* f.read(n) reads the next n characters
(closing and reopening starts from the beginning again)
* reading past the end of a file returns "" (like slicing)
* for line in f: reads a line at a time
* f.readline() reads one line

## processing strings

* strings read from files include \n (newline)
* s.strip() gets rid of newlines and whitespace
* s.split() splits strings into a list (by spaces, by default)
* s.lower(), s.upper() to convert to lower/uppercase
* s.replace(val1,val2) replaces val1 with val2 in s (e.g. cleaning punctuation)

## sets

* collections of objects (any type)
* **unordered** (can’t index or slice), **mutable**
* **iterable**: can use for i in S:, len(), in
* define a new set with {"a","b","c"}; empty set is set(), **NOT** {} (which is a dict) …
* … or convert from a list/tuple/etc. set(["a","b","c"])
* add new elements with S.add("d"). remove with remove()
* **duplicated elements are silently removed**
* .intersection, .union
* .issubset, comparison operators (<, <= etc.)

## dictionaries

* collections of keys and values
* unordered, mutable, iterable
* keys act like a set
* setup via {"A":1, "B":2} or dict([["A",1],["B",2]]) or dict(A=1,B=2)
* keys can be any non-mutable type (int, float, tuple)
* values can be anything
* for i in d: iterates over keys; in searches in keys
* add **or replace** a key/value pair: d[k] = v
* delete a key/value pair: del d[k]
* d[k] extracts the value associated with k
* d.keys() returns keys (set-like); d.values() returns values (list-like); .items() returns a list-like object holding (key,value) tuples
* processing a dictionary (with for k in d: or for k, v in d.items():)
* dictionary inversion

## random number

* random and numpy.random modules (similar)
* random.seed(102): initialize random-number generator (RNG) to a known point (for reproducibility)
* random.randrange(): pick one value from a range
* random.choice(): pick one value from a list/tuple
* random.random(): random float uniformly from $[0,1)$
* random.uniform(a,b): random float uniformly from $[a,b)$

## numpy arrays

* np.array(): from list, tuple, nested lists or tuples
* dtype= argument specifies data type (“float”, “int32”, “int8”, “uint8” etc.)
* a.shape returns a tuple giving dimensions
* len(a) gives length of dimension 0
* also create arrays with np.ones(), np.zeros(), np.arange()
* shape= argument: tuple specifying dimensions; np.ones(4) is the same as np.ones((4,)); np.ones((4,4)) returns a 4 $×$ 4 matrix
* a.fill(v) fills array a with value v

## slicing and indexing arrays

* indexing: a[i] or a[i,j] or a[i,j,k] (depending on dimensions)
* slicing: a[m:n] or a[m:n,:] or …; : by itself means “all rows/columns/slices”
* a.copy() to make a copy

## reshaping arrays

* a.reshape((r,c)) specifies number of columns (total number of elements must match)
* a[:,np.newaxis] adds a new length-1 dimension
* a.flatten() converts to 1-D

## matrices

* np.identity, np.eye for identity matrices
* **not covered**: linear algebra (np.linalg.det, np.linalg.dot, np.linalg.eig, np.linalg.inv)

## operations

* all arithmetic (+, -, \*, etc.) operates **elementwise** on arrays
* … or on array + scalar
* also numpy functions np.sin(), np.cos(), etc.
* np.sum(), np.mean(), np.prod() etc. operate on *all elements* by default
* axis=i argument **collapses dimension i** (e.g. np.mean(a,axis=0) on a 2D array computes mean of each column, collapsing rows)

## logical operations

* comparisons (>, == etc.) work elementwise, producing a bool array
* np.logical\_and(), np.logical\_or(), np.logical\_not()
* a[b] selects the elements of a for which bool array b is True
* e.g. a[a>0] selects positive elements

## numerics

* numpy integers: for an $n$-bit *signed* integer (the default, one bit is used as the sign bit, so the maximum positive value is $2^{n−1}−1$; maximum negative is $−2^{n−1}$
* for an *unsigned* integer (e.g. uint32), the range is from $−2^{n}$ to $2^{n}−1$
* going out of bounds “wraps around”
* plain (not numpy) integers are special, won’t overflow
* floating-point: **often experience rounding error**. Don’t assume math works exactly.
* use np.isclose() or math.isclose() to test near-equality
* overflow
	+ for regular (64-bit) floats, values greater than $≈2^{2^{10}}≈10^{308}$ become inf
	+ values less than $≈−10^{308}$ become -inf
	+ undefined operations (e.g. inf-inf, inf/inf) become nan (not a number)
* underflow
	+ values less than $≈2^{−2^{10}}≈10^{−308}$ become 0
	+ adding *relatively* much smaller numbers (i.e. $a+b$ where $b/a<2^{−53}≈10^{−16}$), they disappear: e.g. $1+x==1$ if $x$ is very small

This appears on the test:

Some helpful numbers: $2^{7}=128$; $2^{8}=256$; $2^{2^{10}}≈10^{308}$; $2^{−53}≈10^{−16}$.

Maybe useful for thinking about integers:

