Master of Engineering in Internetworking

Lab # 4

Lists, Dictionaries, Tuples

INWK 6312

Section A

list: A sequence of values.

element: One of the values in a list (or other sequence), also called items.

index: An integer value that indicates an element in a list.

nested list: A list that is an element of another list.

list traversal: The sequential accessing of each element in a list.

mapping: A relationship in which each element of one set corresponds to an element of another set. For example, a list is a mapping from indices to elements.

accumulator: A variable used in a loop to add up or accumulate a result.

augmented assignment: A statement that updates the value of a variable using an operator like `+=`.

reduce: A processing pattern that traverses a sequence and accumulates the elements into a single result.

map: A processing pattern that traverses a sequence and performs an operation on each element.

filter: A processing pattern that traverses a list and selects the elements that satisfy some criterion.

object: Something a variable can refer to. An object has a type and a value.

equivalent: Having the same value.

identical: Being the same object (which implies equivalence).

reference: The association between a variable and its value.

aliasing: A circumstance where two or more variables refer to the same object.

delimiter: A character or string used to indicate where a string should be split.

!!!!!! STRINGS ARE IMMUTABLE & LIST ARE MUTABLE !!!!!

Common Methods List and Strings Share. They return new value **(they don't change the sequence**)

Operation	Result
x in s	True if an item of *s* is equal to *x*, else False
x not in s	False if an item of *s* is equal to *x*, else True
s + t	the concatenation of *s* and *t*
s * n, n * s	equivalent to adding *s* to itself *n* times
s[i]	*i*th item of *s*, origin 0
s[i:j]	slice of *s* from *i* to *j*
s[i:j:k]	slice of *s* from *i* to *j* with step *k*
len(s)	length of *s*
min(s)	smallest item of *s*
max(s)	largest item of *s*
s.index(x)	index of the first occurrence of *x* in *s*
s.count(x)	total number of occurrences of *x* in *s*

Mutable Sequence Type (List & ByteArray): **(They modify the sequence)**

Operation	Result
s[i] = x	item *i* of *s* is replaced by *x*
s[i:j] = t	slice of *s* from *i* to *j* is replaced by the contents of the iterable *t*
del s[i:j]	same as s[i:j] = []
s[i:j:k] = t	the elements of s[i:j:k] are replaced by those of *t*
del s[i:j:k]	removes the elements of s[i:j:k] from the list
s.append(x)	same as s[len(s):len(s)] = [x]
s.extend(t) or s += t	for the most part the same as s[len(s):len(s)] = t
s *= n	updates *s* with its contents repeated *n* times
s.count(x)	return number of *i*'s for which s[i] == x
s.index(x[, i[, j]])	return smallest *k* such that s[k] == x and i <= k < j
s.insert(i, x)	same as s[i:i] = [x]
s.pop([i])	same as x = s[i]; del s[i]; return x
s.remove(x)	same as del s[s.index(x)]

Operation	Result
s.reverse()	reverses the items of *s* in place
s.sort([cmp[, key[, reverse]]])	sort the items of *s* in place

REDUCE:

A processing pattern that traverses a sequence and accumulates the elements into a single result.

```
def add_all(t):
    total = 0 #Accumulator
     for x in t:
        total += x
     return total
```

Exercise

Write a function called `nested_sum` *that takes a nested list of integers and add up the elements from all of the nested lists.*

BONUS:

Built-in Function: Reduce -- *read documentation about it*

`reduce`(*function*, *iterable*[, *initializer*])

Question:

Using the built-in reduce function, find the sum of even numbers between 100 and 500

MAP:

A processing pattern that traverses a sequence and performs an operation on each element.

```
def capitalize_all(t):
     res = []
     for s in t:
       res.append(s.capitalize())
    return res
```

BONUS:

Built-in Function: Map -- *read documentation about it*

map(*function*, *iterable*, ...)

Apply *function* to every item of *iterable* and return a list of the results. If additional *iterable* arguments are passed, *function* must take that many arguments and is applied to the items from all iterables in parallel.

Question

Using the built-in reduce function, find the sum of the square of even numbers between 0 and 10

FILTER :

A processing pattern that traverses a list and selects the elements that satisfy some criterion.

```
def only_upper(t):
     res = []
     for s in t:
       if s.isupper():
          res.append(s)
    return res
```

BONUS:

Built-in Function: filter -- *read documentation about it*

`filter`(*function*, *iterable*)

Construct a list from those elements of *iterable* for which *function* returns true. *iterable* may be either a sequence, a container which supports iteration, or an iterator.

Question

Given a list of number, return a list of the numbers that are multiples of 5

Question 1

Use `capitalize_all` *to write a function named* `capitalize_nested` *that takes a nested list of strings and returns a new nested list with all strings capitalized.*

Question 2

Write a function that takes a list of numbers and returns the cumulative sum; that is, a new list where the ith element is the sum of the first i+1 elements from the original list. For example, the cumulative sum of `[1, 2, 3]` *is* `[1, 3, 6]`.

Question 3

Write a function called `middle` *that takes a list and returns a new list that contains all but the first and last elements. So* `middle([1,2,3,4])` *should return* `[2,3]`.

Question 4

Write a function called `is_sorted` *that takes a list as a parameter and returns* `True` *if the list is sorted in ascending order and* `False` *otherwise. You can assume (as a precondition) that the elements of the list can be compared with the relational operators* `<`, `>`, *etc.*
For example, `is_sorted([1,2,2])` *should return* `True` *and* `is_sorted(['b','a'])` *should return* `False`.

Question 5

Two words are anagrams if you can rearrange the letters from one to spell the other. Write a function called `is_anagram` *that takes two strings and returns* `True`. *if they are anagrams.*

Question 6

The (so-called) Birthday Paradox:

1. *Write a function called* `has_duplicates` *that takes a list and returns* `True` *if there is any element that appears more than once. It should not modify the original list.*
2. *If there are 23 students in your class, what are the chances that two of you have the same birthday? You can estimate this probability by generating random samples of 23 birthdays and checking for matches. Hint: you can generate random birthdays with the* `randint` *function in the* `random` *module.*

Question 7

Write a function called `remove_duplicates` *that takes a list and returns a new list with only the unique elements from the original. Hint: they don't have to be in the same order.*

Section B

dictionary: A mapping from a set of keys to their corresponding values.

key-value pair: The representation of the mapping from a key to a value.

item: Another name for a key-value pair.

key: An object that appears in a dictionary as the first part of a key-value pair.

value: An object that appears in a dictionary as the second part of a key-value pair. This is more specific than our previous use of the word "value."

implementation: A way of performing a computation.

hashtable: The algorithm used to implement Python dictionaries.

hash function: A function used by a hashtable to compute the location for a key.

hashable: A type that has a hash function. Immutable types like integers, floats and strings are hashable; mutable types like lists and dictionaries are not.

lookup: A dictionary operation that takes a key and finds the corresponding value.

reverse lookup: A dictionary operation that takes a value and finds one or more keys that map to it.

singleton: A list (or other sequence) with a single element.

call graph: A diagram that shows every frame created during the execution of a program, with an arrow from each caller to each callee.

histogram: A set of counters.

memo: A computed value stored to avoid unnecessary future computation.

flag:A boolean variable used to indicate whether a condition is true.

declaration: A statement like `global` that tells the interpreter something about a variable.

REFERENCES:

Mapping Operations

https://docs.python.org/2.7/library/stdtypes.html?highlight=get - mapping-types-dict.

Looping Techniques:

https://docs.python.org/2.7/tutorial/datastructures.html?highlight=dictionary

Raising Exceptions and Built-in Exceptions:

https://docs.python.org/2.7/library/exceptions.html#module-exceptions

https://docs.python.org/2.7/tutorial/errors.html?highlight=raise%20exception#raising-exceptions

Question 1

Write a function that reads the words in `words.txt` *and stores them as keys in a dictionary. It doesn't matter what the values are. Then you can use the* `in` *operator as a fast way to check whether a string is in the dictionary.*

Question 2

Dictionaries have a method called `get` *that takes a key and a default value. If the key appears in the dictionary,* `get` *returns the corresponding value; otherwise it returns the default value. For example:*

```
>>> h = histogram('a')
>>> print h
{'a': 1}
>>> h.get('a', 0)
1
>>> h.get('b', 0)
0
```

Use `get` *to write* `histogram` *more concisely. You should be able to eliminate the* `if` *statement.*

Question 3

Dictionaries have a method called `keys` *that returns the keys of the dictionary, in no particular order, as a list.*

```
def print_hist(h):
    for c in h:
        print c, h[c]
```

Modify `print_hist` *to print the keys and their values in alphabetical order.*

Question 4

Modify `reverse_lookup` *so that it builds and returns a list of* all *keys that map to* `v`*, or an empty list if there are none*.

```
def reverse_lookup(d, v):
    for k in d:
        if d[k] == v:
            return k
    raise ValueError
```

Question 5

Read the documentation of the dictionary method `setdefault` *and use it to write a more concise version of* `invert_dict`*.*

```
def invert_dict(d):
    inverse = dict()
    for key in d:
        val = d[key]
        if val not in inverse:
            inverse[val] = [key]
        else:
            inverse[val].append(key)
    return inverse
```

Question 6

Run this version of `fibonacci` *and the original with a range of parameters and compare their run times.*

```
known = {0:0, 1:1}

def fibonacci(n):
    if n in known:
        return known[n]

    res = fibonacci(n-1) + fibonacci(n-2)
    known[n] = res
    return res
```

Question 7

Below there is a function named `has_duplicates` *that takes a list as a parameter and returns* `True` *if there is any object that appears more than once in the list.*

```
def has_duplicates(t):
    d = {}
    for x in t:
        if x in d:
            return True
        d[x] = True
    return False
```

Use a dictionary to write a faster, simpler version of `has_duplicates`

SECTION C

tuple: An immutable sequence of elements.

tuple assignment: An assignment with a sequence on the right side and a tuple of variables on the left. The right side is evaluated and then its elements are assigned to the variables on the left.

gather: The operation of assembling a variable-length argument tuple.

scatter: The operation of treating a sequence as a list of arguments.

DSU: Abbreviation of "decorate-sort-undecorate," a pattern that involves building a list of tuples, sorting, and extracting part of the result.

data structure: A collection of related values, often organized in lists, dictionaries, tuples, etc.

shape (of a data structure): A summary of the type, size and composition of a data structure.

Question 1

Many of the built-in functions use variable-length argument tuples. For example, `max` and `min` can take any number of arguments:

```
>>> max(1,2,3)
3
```

But `sum` does not.

```
>>> sum(1,2,3)
TypeError: sum expected at most 2 arguments, got 3
```

Write a function called `sumall` that takes any number of arguments and returns their sum.

Question 2

In this example, ties are broken by comparing words, so words with the same length appear in reverse alphabetical order. For other applications, you might want to break ties at random. Modify this example so that words with the same length appear in random order

```
def sort_by_length(words):
    t = []
    for word in words:
       t.append((len(word), word))

    t.sort(reverse=True)

    res = []
    for length, word in t:
        res.append(word)
    return res
```

Question 3

Write a function called most frequent that takes a string and prints the letters in decreasing order of frequency.

Question 4

1. *Write a program that reads a word list from a file (see Section 9.1) and prints all the sets of words that are anagrams.*

   Here is an example of what the output might look like:

   ```
   ['deltas', 'desalt', 'lasted', 'salted', 'slated', 'staled']
   ['retainers', 'ternaries']
   ['generating', 'greatening']
   ['resmelts', 'smelters', 'termless']
   ```

   Hint: you might want to build a dictionary that maps from a set of letters to a list of words that can be spelled with those letters. The question is, how can you represent the set of letters in a way that can be used as a key?

2. *Modify the previous program so that it prints the largest set of anagrams first, followed by the second largest set, and so on.*