LECTURE 4: PYTHON DATA STRUCTURES AND FILES

ANNOUNCEMENT

- Ph.D. TAs:
  - Ying Lu (Office hours: TBD)
  - David Paredes (Office hours: TBD)

FIXED SIZE AND NESTED ARRAYS

```
>>> data = [0]*5

>>> data

[0, 0, 0, 0, 0]

>>> for i in range(5): data[i] = [0]*5

>>> data

>>> [[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0,
0, 0], [0, 0, 0, 0, 0]]
```

LIST ASSIGNMENTS AND REFERENCING

```
>>> a = [1,2,3]
>>> b = a
>>> b[1] = 7
>>> a
[1, 7, 3]
```

- Use true copy!

```
>>> a = [1,2,3]
>>> b = a[:]
```

IDENTITY AND EQUALITY

- == means equality (Are the values equal?)
- “is” means identity (Are the objects identical(i.e., points to same memory location?)
- Two are different
- “x == y” could be True, while "x is y” is False
- Example: Lists vs Strings
- Strings are *interned*

STRING EXAMPLE: PALINDROME

```
def pal_test(string):
    if len(string) <= 1:
        return True
    elif string[0] != string[-1]:
        return False
    else:
        return pal_test(string[1:-1])
```

STRING OPERATIONS

Operation	Description
s.capitalize()	Capitalizes the first character of s
s.capwords()	Capitalizes the fist letter of each word in s
s.count(sub)	Counts the number of occurrences of sub in s
s.find(sub)	Finds the first index of sub in s, or -1 if not found
s.index(sub)	Finds the first index of sub in s, or raise a ValueError if not found
s.rfind(sub)	Finds the last index of sub in s, or -1 if not found
s.rindex(sub)	Finds the last index of sub in s, or raise a ValueError if not found
s.lower()	Converts s to lowercase
s.upper()	Converts s to upper case
s.split(sep)	Using sep as the separator, returns a list of words in string s
s.join(lst)	Joins a list of words into a single string with s as separator
s.strip()	Strips leading/trailing white space from s
s.replace(old, new)	Replaces all instances of old with new in string

JOIN AND SPLIT

- Join: From list to string
- Split: From string to list
- Application: Count the words in a string

```
>>> string = ”to be or not to be”
>>> string.split(" ")
['to', 'be', 'or', 'not’, 'to', 'be'
>>> len(string.split(" "))
6
```

TUPLES

- Very similar to lists, but **immutable** (Can’t change, add or remove elements)
- Type conversion with tuple()
- Use cases?
  - Multiple return values
  - Simultaneous assignments
- Advantages?

DICTIONARIES

- Unordered key, value pairs (Keys are unique)
- Indexing by keys(vs numbers in lists)
- Two ways to define an empty dictionary
  - dict()
  - {}
- Assignments:

```
>>> dct = {}
>>> dct[‘name’] = “ali”
>>> dct[‘pet’] = “cat”
```

NON-EMPTY INITIALIZATION

```
>>> phone_numbers = {'john': '555-555-55-55', 'mary': '444-444-44-44', 'chris':
'333-333-33-33'}
>>> phone_numbers
{'chris': '333-333-33-33', 'john': '555-555-55-55', 'mary': '444-444-44-44'}
>>> phone_numbers.keys()
['chris', 'john', 'mary’]
>>> phone_numbers.values()
['333-333-33-33', '555-555-55-55', '444-444-44-44’]
>>> 'john' in phone_numbers
True
```

DICTIONARY OPERATIONS

Operation	**Description - Giving a dictionary d,**
len(d)	Number of items in d
d[k]	Item in d with key k
d[k] = v	Set item in d with key k to have value v
del d[k]	Delete item k from dictionary d
d.clear()	Remove all items from dictionary d
d.copy()	Make a shallow copy of d (see slide "Making Copies")
d.has_key(k)	Return 1 if d has key k, 0 otherwise
d.items()	Return a list of (key,value) pairs
d.keys()	Return a list of keys in d
d.values()	Return a list of values in d
d.get(k)	Same as d[k]
d.get(k,v)	Return d[k] if k is valid, otherwise return v

EXAMPLE: COUNTING ELEMENTS

```
def frequency(lst):
    counts = {}
    for ele in lst:
        counts[ele] = counts.get(ele,0) + 1
    return counts
>>> frequency(['abc', 'def', 'abc', 'pdq', 'abc’])
{'abc': 3, 'pdq': 1, 'def': 1}
>>> frequency('the best of the best'.split())
{'of': 1, 'the': 2, 'best': 2}
```

UPDATING/COPYING

- Updating: dict1.update(dict2)
- It may overwrite existing entries!
- Copying: dict2 = dict1.copy()

FILE OPERATIONS

Operation	Description
f=open("filename")	Open a file, return a file value
f=open("filename","w")	Open a file for writing, overwrites it if file exists
f.read()	Return the entire file contents to a string
f.read(n)	Return no more than n character values
f.readline()	Return the next line of input
f.readlines()	Return all the file as a list
f.write(s)	Write string s to file
f.writelines(lst)	Write list lst to file
f.close()	Close the file

SAMPLE FILE OPERATIONS

- See peas.py and peas.txt

FOR LOOPS WITH FILES

```
f = open('peas.txt’)
counts = {}
for eachline in f:
    for ele in eachline.split():
        counts[ele] = counts.get(ele,0) + 1
        print counts
f.close()
```

EXCEPTION HANDLING

```
try:
    f=open("nonexist.txt")
except IOError:
    print('Unable to open the file nonexist.txt’)
else:# if try does not raise an exception
    f.read(1024)
    f.close()
    print('Read.’)
print('Done.’)
```

FUNCTIONAL PROGRAMMING

- Not interested in the concept, but in the application
  - Lambda
  - Map, filter, reduce
  - List comprehension
- Try to avoid mutable data, treat computations like mathematical functions

LAMBDA FUNCTIONS

```
>>> square = lambda x: x**2
>>> square(2)
4

>>> add = lambda x,y : x + y
>>> add(5,3)
8

>>> lp = lambda: print("hello") # works in python 3
>>> lp()
hello
```

MAPPING, FILTERING, REDUCTION

- Mapping: One-to-one transformation through a function
  - [1,2,3,4,5] -> [1,4,9,16,25] (lambda x: x**2)
- Filtering: Apply a condition, retain ones that satisfy the condition
  - [1,2,3,4,5] -> [2,4] (lambda x: x % 2 == 0)
- Reduction: Apply a binary function to each member of a list in a row
  - [1,2,3,4,5] -> 15 (lambda x,y: x + y)
  - (((1+2)+3)+4)+5=15
- Reserved keywords: filter, map, reduce

EXAMPLES

```
>>> lst = [1,2,3,4,5]
>>> list(map(lambda x: x ** 2,lst))
[1, 4, 9, 16, 25]
>>> filter(lambda x:  x % 2 == 0,lst)
[2, 4]
>>> functools.reduce(lambda x,y: x + y,lst)
15
>>> list(map(lambda x: 1 if x % 2 == 0 else 0,lst))
[0, 1, 0, 1, 0]
```

REDUCE IN PYTHON3

- Import functools first
  - import functools

EXERCISE 1: FIND VALUES FOR KEYS

```
>>> c={'a':1,'b':2,'c':3}
>>> keys = ['a','b','c']
>>> list(map(lambda x: c[x],['a','b','c’])) # method 1
[1, 2, 3]
>>> list(map(lambda x,y: y[x],['a','b','c'],[c,c,c])) # method 2
[1, 2, 3]
```

FEATURES OF THE RESULTS

- The original list remains unchanged
- The three functions map, filter, and reduce produce new lists that are transformation of the argument
- A function that uses another function that is passed as an argument is referred to as a higher order function

LIST COMPREHENSION

```
>>> lst = [1,2,3,4,5]
>>> [x**2 for x in lst]
[1, 4, 9, 16, 25]

>>> [x**2 for x in lst if x%2 ==0]
[4, 16]

>>> [1 if x%2==0 else 0 for x in lst]
[0, 1, 0, 1, 0]
```

FIN