APS106

UNIVERSITY OF TORONTO

Tutorial 8 – Week 9

We'll be starting at the 10 minute mark

if nothing else, write #cleancode

Agenda

- Lab review
  - More practice with iteration
- Lecture review – data structures
  - tuples
  - sets
  - dictionaries
- Practice questions

Learning Objectives

After this tutorial, learners should be able to:

- recognize / describe / create objects of type tuple/set/dictionary
- access and modify individual elements of a tuple using subscription and slicing
- access and modify individual elements of a dictionary using subscription
- insert and remove elements from a tuple/set/dictionary using appropriate tuple/set/dictionary methods/operators
- test whether a value/object belongs to a tuple/set/dictionary using the membership operator
- iterate over tuples/sets/dictionaries

Lab review

More practice with iteration & Strings

Lab 5 review[1]

```
def email_to_name(email):
    """
    (str) -> str

    Given a string with the format "first_name.last_name@domain.com",
    return a string "LAST_NAME,FIRST_NAME" where all the characters are upper
    case

    >>> email_to_name("anna.conda@mail.utoronto.ca")
    'CONDA,ANNA'
    """
```

use `.find()` to obtain the position of “.” and then use string slicing to obtain `first_name`

use `.find()` to obtain the position of “@” and then use string slicing to obtain `last_name`.

This can be achieved by string concatenation

Lab 5 review^2

```
def count_measurements(s):
    """
    (str) -> int

    Given s, a string representation of comma separated site-measurement
    pairs, return the total number of measurements

    >>> count_measurements("B, 5.6, Control, 5.5, Db, 3.2")
    3

    >>> count_measurements("Control, 7.5")
    1
    """
```

Split the string up using “,” as a separator, and then count the resulting elements. The number of measurement is the number of elements divided by 2.

Alternatively, you can count the number of commas using `.count()` and determine the number of measurements based on the number of commas.

Lab 5 review^3

```
def calc_site_average(measurements, site):
    """
    (str, str) -> float

    Given s, a string representation of comma separated site-measurement
    pairs, return the average of the site measurements to one
    decimal place

    >>> calc_site_average("A, 4.2, B, 6.7, Control, 7.1, B, 6.5, Control,
  7.8, Control, 6.8, A, 3.9", "Control")

    7.2
    """
```

You can use `.find(value, start, end)` to identify the index of `site`. Multiple occurrences of `site` may be present. You can use a loop to find all occurrences.

Hint: specify a different start index in `.find(value, start, end)` in each iteration.

Alternatively, use string slicing to truncate the string in each iteration to remove the part of the string that has already been processed.

Review of Lecture

Tuples

Tuples[1]

- A tuple is an **ordered sequence** of items, similar to lists, but with some key differences:
  - Tuples are written using parentheses ( )
  - Tuples are **immutable**

- Tuples can be indexed or sliced (like lists and strings)

- Tuples elements CANNOT be replaced

```
>>> class_score = (['sara', 75.0], ['cris', 72.5])
>>> class_score[0] = class_score[1]
```

This assignment will result in an error since tuple objects are immutable

- Tuples elements CAN be modified

```
>>> class_score[0][1] = 80.2
>>> class_score[0]
  ['sara', 80.2]
```

This assignment does not raise an error because list objects are mutable

Tuples^2

- Tuples can be used to swap the values of multiple variables in just one line of code
- Let’s say we want to swap the values of variable `a` and `b`:

General strategy

```
>>> a = 1
>>> b = 2
>>> temp = a
>>> a = b
>>> b = temp
>>> print("a =", a)
   a = 2
>>> print("b =", b)
   b = 1
```

Using tuple

```
>>> a = 1
>>> b = 2
>>> (a, b) = (b, a)
>>> print("a =", a)
   a = 2
>>> print("b =", b)
    b = 1
```

Tuples^{3}

Want to return multiple variables from a function?

⇒ Return them packed in a tuple object.

```
def trig_calculator(deg):
    '''
    (float) -> (float)
    calculates trig functions given an angle
    '''

    sin = math.sin(math.radians(deg))
    cos = math.cos(math.radians(deg))
    tan = math.tan(math.radians(deg))

    return (sin, cos, tan)
```

```
>>> returned_values = trig_calculator(deg)
>>> print(type(returned_values))
<class 'tuple'>
```

Tuple Operations – as Operators

Operation	Operator	Example

Indexing	[ ]	`>>> a = (20 ,40, "apple", "ball")`
		`>>> a[0]`
		`20`

Slicing	[: :]	`>>> a[1:3]`
		`(40, 60)`

Concatenation	+	`>>> c = (1, )`
		`>>> b = (2, 4)`
		`>>> c + b`
		`(1, 2, 4)`

Repetition		`>>> b * 2`
		`(2, 4, 2, 4)`

Membership		`>>> 20 in a`
		`True`
		`>>> "orange" in a`
		`False`

Comparison		`>>> d = (2,3,4)`
		`>>> a == b`
		`False`

Tuple Operations – as Methods

Operation	Methods	Example

Find the index of the first occurrence of a given value in a tuple.	`index(value)`	`>>> b = (1,2,3,2,1)`
		`>>> b.index(2)`
		`1`

Find the number of times a given value appears in a tuple.	`count(value)`	`>>> b = (1,2,3,2,1)`
		`>>> b.count(2)`
		`2`

Built-in Functions that can be used on Tuples

- `min()`: return the minimum element in a tuple
- `max()`: return the maximum element in a tuple
- `len()`: return the length of the tuple

Review of Lecture

Sets

Sets

- A set is an **unordered** collection of **distinct** items
- The set notation is similar to lists and tuples, but uses curly brackets **{ }**
- Set elements do not have a fixed position in the set
  ⇒ We cannot use indexing

```
>>> vowels = set()

>>> vowels = {'a', 'a', 'e', 'i', 'o', 'u'}
>>> vowels
    {'a', 'u', 'o', 'i', 'e'}

>>> vowels
    {'o', 'u', 'a', 'i', 'e'}

>>> {1, 2} == {2, 1}
   True
>>> [1, 2] == [2, 1]
   False
>>> (1, 2) == (2, 1)
   False
```

An empty set

Duplicates are removed.

The items in the set may have a different order.

Sets Operations

Just as in set theory we can perform common mathematical operations on sets.

Operation	Equiv.	Description
len(s)		number of elements in set s
x in s		test x for membership in s
x not in s		test x for non-membership in s
s.issubset(t)	s <= t	test whether every element in s is in t
s.issuperset(t)	s >= t	test whether every element in t is in s
s.union(t)	s | t	new set with elements from both s and t
s.intersection(t)	s & t	new set with elements common to s and t
s.difference(t)	s - t	new set with elements in s but not in t
s.symmetric_difference(t)	s ^ t	new set with elements in either s or t but not both
s.copy()		new set with a copy of s

Review of Lecture

Dictionaries

Dictionaries

- A dictionary is a collection of unordered, distinct key-value pairs

- A common way to create dictionaries is to use curly braces, **{}**, around key-value pairs of literals and/or values expressed as expressions
    - The general syntax is: { key1 : expr1, key2 : expr2, …, keyN : exprN }
    - The `key : value` pairs of the dictionary are separated by commas.
    - Each pair contains a key (always a literal) and a value separated by a colon.

- Unlike sequences, which are indexed by a range of numbers, dictionaries are indexed by ***keys***.
    - Keys can be values of any immutable type, e.g., strings, numbers, etc.
        - Tuples can be used as keys if they contain only immutable elements;
            - If a tuple contains any mutable object either directly or indirectly, it cannot be used as a key.

Dictionary operations

- Given: `d1 = {'Tina':'A+', 'Min':'A'}`

Operation	Operator / Method	Example code

Indexing: retrieve the value associated with a key.	[ ]	`>>> d1['Tina']`
		`'A+'`

Add an entry if the entry does not exist, otherwise modify the existing entry	[ ]	`>>> d1['John'] = 'B+'`
		`>>> d1`
		`{'John': 'B+', 'Tina': 'A+', 'Min':'A'}`

Delete/remove a given key and its associated value from a dictionary	del, pop	`>>> del d1['Tina']`
		`>>> d1`
		`{'John': 'B+', 'Min':'A'}`
		`>>> d1.pop('John')`
		`>>> d1`
		`>>> {'Min':'A'}`

Test if a given key is in the dictionary (it does not check the values)		`>>> 'John' in d1`
		`True`
		`>>> 'B+' in d1`
		`False`

Review of Lecture

Summary

	Ordered	Mutable	Iterable
List	✓	✓	✓
Tuple	✓		✓
Set		✓	✓
Dictionary		✓	✓
String	✓		✓

Operation Example	Result	String	List	Tuple	Set	Dictionary
`x in s`	`True` if an item of *s* is equal to *x*, else `False`	✓	✓	✓	✓	✓ (for keys only)
`x not in s`	`False` if an item of *s* is equal to *x*, else `True`	✓	✓	✓	✓	✓ (for keys only)
`s + t`	the concatenation of *s* and *t*	✓	✓	✓		
`s * n` or `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*	✓	✓	✓		
`s1 == s2`	Equality comparison	✓	✓	✓	✓	✓
`>`, `>=`, `<`, `<=`	Lexicographical comparison (* subset testing for `set` objects)	✓	✓	✓	✓*	
`s[i] = x`	item *i* of *s* is replaced by *x*		✓			✓ (possible if `i` is one of the keys, or if key `i` does not exist)
`del s[i]`	removes the element `s[i]` from the iterable		✓			✓ (possible if `i` is one of the keys)
`s[i:j] = t`	slice of *s* from *i* to *j* is replaced by the contents of the iterable *t*		✓			
`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 the iterable. (`s[i:j]`is the same as `s[i:j] = []`)		✓			
`s *= n`	updates *s* with its contents repeated *n* times	✓	✓	✓		

Operation Example	Result	String	List	Tuple	Set	Dictionary
`s.count(x)`	total number of occurrences of *x* in *s*	✓	✓	✓		
`s.append(x)`	appends *x* to the end of the sequence (same as `s[len(s):len(s)] = [x]`)		✓			
`s.extend(t)` or `s += t`	extends *s* with the contents of *t* (for the most part the same as `s[len(s):len(s)] = t`)		✓			
`s.index(x[, i[, j]])`	index of the first occurrence of *x* in *s* (at or after index *i* and before index *j*)	✓	✓	✓		
`s.insert(i, x)`	inserts *x* into *s* at the index given by *i* (same as `s[i:i] = [x]`)		✓			
`s.pop()` or `s.pop(i)`	retrieves the item at *i* and also removes it from *s*		✓		✓ (`s.pop()` removes a random element from the set. does not support `s.pop(i)`)	✓ (does not support `s.pop()`, and `i` must be one of the keys)
`s.remove(x)`	remove the first item x from *s*		✓		✓	

Summary – Built-in functions

Operation	Result	String	List	Tuple	Set	Dictionary
`len(s)`	length of *s*	✓	✓	✓	✓	✓
`min(s)`	smallest item of *s*	✓	✓	✓	✓	✓ (for keys only)
`max(s)`	largest item of *s*	✓	✓	✓	✓	✓ (for keys only)

Practice Problems

Coding Question 1

Write a function `combine()` that combines values from an input list of dictionaries of type `{'item': i1, 'amount': a1 }`. `combine()` should return a dictionary whose keys are the values associated with the keys `item` in the input dictionaries. The value associated with each key is the sum of the amounts corresponding to that key in the input dictionaries.

Usage example:

```
>>> result = combine( [{'item': 'i1', 'amount': 400}, {
    {'item': 'i2', 'amount': 300}, {'item': 'i1', 'amount′: 750}])

>>> result
    {'i1': 1150, 'i2': 300}
```

Coding Question 2

Complete the function body below according to its docstring description.

```
def get_student_class(student_to_class, myclass):
    """
    (dict, list[str]) -> list[str]
    In student_to_class each key is a student and each value is the
    lists of classes each student is enrolled in. myclass is a list of
    classes given as strings.Return a list of students from
    student_to_class who are in myclass.
    >>> D = {‘Ana’: [‘csc258’, ‘aps106’, ‘mat188’],‘Frank’: [‘aps106’,
    ‘mat182’], ‘Rachel’: [‘mat188’]}
    >>> get_student_class(D, ‘aps106’)
        [‘Ana’, ‘Frank’]
    """
```

slido

Any Questions?

ⓘ Start presenting to display the poll results on this slide.