STOP 0 – Mutable Data Types

- <list>.copy(), <dict>.copy() and <set>.copy()

```
1 var_l1 = [1,2,3] # [1,2,3]
2 var_l2 = var_l1 # [1,2,3]
3 var_l3 = var_l1.copy() # [1,2,3]
4 var_l1.append("NEW") #
5 var_l1 # [1, 2, 3, 'NEW']
6 var_l2 # [1, 2, 3, 'NEW']
7 var_l3 # [1, 2, 3]
```

- Zelle
  - Page 153-154
    - addinterest2.py and addinterest3.py

STOP 0 – Dictionaries

- Quick recap

```
1 var_dict1 = {True: 'Will this work?'} # Okay!
2 var_dict2 = {0.59: 'Will this work?'} # Okay!
3 var_dict3 = {{1: 'one'}: 'How about this?'} #
Error! Key is mutable
4 var_dict4 = {'key': {1: 'one'}} # Okay!
5
6 var_dict5 = {'key': [1, 2, 3]} # Okay!
7 var_dict6 = {[1, 2, 3]: 'key'} # Error! Key is
mutable!
```

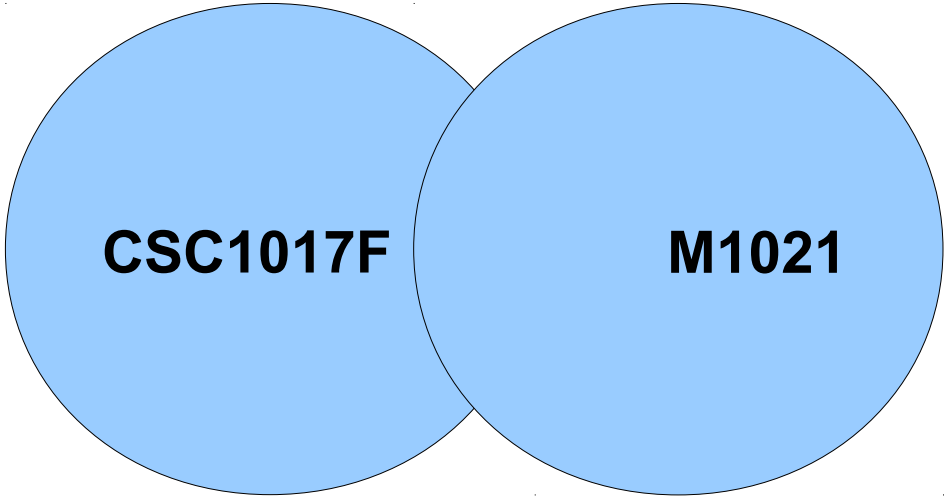

UCT Department of Computer Science
Computer Science 1017F

Sets& Nested Collections

Lighton Phiri <lphiri@cs.uct.ac.za>
April 2015

Practical Applications

- Sets are Python's implementation of mathematical set theory
- What if we pulled another Vula course database and wanted to figure out how many of us in here are also enrolled for that course?



Introduction

- Sets are used to unique values
  - Duplicate entries are merged/eliminated
- Sets are defined using
  - set() for empty or non-empty set definition
  - {} for non-empty set definition
- Values are separated by commas
- Values can be of any immutable data type
- Sets are mutable

Creating Sets

```
1 var_set = {1, 2, 3}
2 type (var_set) # <class 'set'>
3
4 var_another_set = set() # empty set
5 type (var_another_set) # <class 'set'>
6
7 var_what = {} # {}
8 type (var_what) # <class 'dict'>
```

- Values separated by commas
- Notice that Line 7 results in the creation of a dictionary NOT a set

Sets in Pespective

- Sets are used to unique values
  - Duplicate entries are merged/eliminated

```
1 var_set = {1, 2, 3, 1, 3} # {1, 2, 3}
#
```

- Values can be of any immutable data type

```
1 var_set1 = {True, 1, False} #  {False, 1}
2 var_set2 = {[1, 2, 3]} # Error!
3 var_set3 = {{'x':1, 'y':2, 'z':3}} # Error!
```


Sets in Pespective

- Sets are mutable
  - Set methods can be used to perform CRUD actions of set members

```
1 var_set = {1, 2, 3}
2
3 var_set.add('a') # {1, 2, 3, 'a'}
4 var_set.remove('a')  # {1, 2, 3}
5 var_set.clear(); # set()
```


STOP 1 – Questions

STOP 2 – Methods vs Functions

```
1 import math
2
3 def add (a, b):
4    return a+b
5
6 math.pow(10, 2) # function invocation from module
7 add (1, 1) # standalone function invocation
8 student_name = “Singh, Shekhar”
9 student_name.upper () # method call—class function invocation
```

- Set **methods** are invocked on instances of set using “.” operator

Set Methods

- <set>.intersection(s)
  - Returns a new set with members in first and second sets
  - Parameter(s): set; Return value: set

```
1 var_set1 = {1, 2, 3}
2 var_set2 = {2, 4, 6}
3 var_set3 = var_set1.intersection(var_set2) #
4
5 var_set3 # {2}
6 type(var_set3) # <class 'set'>
```


Set Methods

- <set>.intersection_update(s)
  - Returns a modified version of the first set
  - Parameter(s): set; Return value: None

```
1 var_set1 = {1, 2, 3}
2 var_set2 = {2, 4, 6}
3 var_set3 = var_set1.intersection(var_set2) #
4
5 type(var_set3) # <class 'set'>
```


More Set Methods

- Poke into additional methods using the help function
  - Pay particular attention to parameters and return types

```
1 help(set)
2 |   add(...)
3 |   clear(...)
4 |   copy(...)
5 |   difference(...)
6 |   difference_update(...)
7 |   discard(...)
8 |   union(...)
9   :
10  :
```


Two-dimensional Lists

1	2	3
“a”	“b”	“c”
2	4	6

- Nested lists can used to create two-dimensional data structures—synonymous to a matrix or grid

```
1 var_nest = [[1,2,3], [“a”,“b”,“c”], [2,4,6]]
```

- Each row is represented by an internal list and referenced by a single index

```
1 var_nest[1] # [“a”,“b”,“c”]
```

- Each item is referenced by two indices

```
1 var_nest[1][-1] # [“c”]
```


Multi-dimensional Lists

	Col 0	Col 1	Col 2
Row 0	1	2	3
Row 1	“a”	“b”	“c”
Row 2	2	4	6

- Nested lists can additionally be used to create multi-dimensional data structures

```
1 var_nest = [[[1,2],[2,4]], [[“a”,“b”],[“c”, “d”]]]
2 var_nest[0][1][1] # 4
```


More Nested Collections

- Certain use cases require manipulation of nested collection structures
  - Reconfiguring the CSC1017F Vula database

```
{
  "students": {
      53: {'name': 'Dlamini, Thandolwethu', 'id': 'dlmtha028', 'email': 'DLMTHA028@myuct.ac.za', 'role': 'Student'},
      88: {'name': 'Lie, Angel', 'id': 'lxxhsi006', 'email': 'LXXHSI006@myuct.ac.za', 'role': 'Student'}
  }
}
```

STOP 3 – Nested Collections

- Are the following possible?
  - Lists in Lists; Sets in Sets; Sets in Lists; Dictionaries in Dictionaries; Sets in Dictionaries
- How can we access name and email for record 53?

```
{
  "students": {
      53: {'name': 'Dlamini, Thandolwethu', 'id': 'dlmtha028', 'email': 'DLMTHA028@myuct.ac.za', 'role': 'Student'},
      88: {'name': 'Lie, Angel', 'id': 'lxxhsi006', 'email': 'LXXHSI006@myuct.ac.za', 'role': 'Student'}
  }
}
```

STOP 4 – Collection Conversion

```
1 from csc1017fvuladb import csc1017f_vula_list
2 var_vula = csc1017f_vula_list()
3 type (var_vula) # <class 'list'>
4
5 var_vula[10] # 'Xx, Yy;XY;XY@myuct.ac.za;Observer'
6
7 [
['Dlamini, Thandolwethu', 'dlmtha028', 'DLMTHA028@myuct.ac.za', 'Student'],
['Naidoo, Kimeshan', 'ndxkim024', 'NDXKIM024@myuct.ac.za', 'Student'],
['Lie, Angel', 'lxxhsi006', 'LXXHSI006@myuct.ac.za', 'Student'],
['Van Zyl, Jayd', 'vzyjoh019', 'VZYJOH019@myuct.ac.za', 'Student']
]
```


STOP 4 – Collection Conversion

```
[
{'name': 'Dlamini, Thandolwethu', 'id': 'dlmtha028', 'email':
'DLMTHA028@myuct.ac.za', 'role': 'Student'},
{'name': 'Naidoo, Kimeshan', 'id': 'ndxkim024', 'email':
'NDXKIM024@myuct.ac.za', 'role': 'Student'},
{'name': 'Lie, Angel', 'id': 'lxxhsi006', 'email': 'LXXHSI006@myuct.ac.za',
'role': 'Student'},
{'name': 'Van Zyl, Jayd', 'id': 'vzyjoh019', 'email':
'VZYJOH019@myuct.ac.za', 'role': 'Student'}
]
```

```
[
{'Dlamini, Thandolwethu', 'dlmtha028', 'DLMTHA028@myuct.ac.za', 'Student'},
{'Naidoo, Kimeshan', 'ndxkim024', 'NDXKIM024@myuct.ac.za', 'Student'},
{'Lie, Angel', 'lxxhsi006', 'LXXHSI006@myuct.ac.za', 'Student'},
{'Van Zyl, Jayd', 'vzyjoh019', 'VZYJOH019@myuct.ac.za', 'Student'}
]
```


STOP 4 – Collection Conversion

```
{
53: {'Dlamini, Thandolwethu', 'dlmtha028', 'DLMTHA028@myuct.ac.za',
'Student'},
120: {'Naidoo, Kimeshan', 'ndxkim024', 'NDXKIM024@myuct.ac.za', 'Student'},
88: {'Lie, Angel', 'lxxhsi006', 'LXXHSI006@myuct.ac.za', 'Student'},
178: {'Van Zyl, Jayd', 'vzyjoh019', 'VZYJOH019@myuct.ac.za', 'Student'}
}


{
53: {'name': 'Dlamini, Thandolwethu', 'id': 'dlmtha028', 'email':
'DLMTHA028@myuct.ac.za', 'role': 'Student'},
120: {'name': 'Naidoo, Kimeshan', 'id': 'ndxkim024', 'email':
'NDXKIM024@myuct.ac.za', 'role': 'Student'},
88: {'name': 'Lie, Angel', 'id': 'lxxhsi006', 'email':
'LXXHSI006@myuct.ac.za', 'role': 'Student'},
178: {'name': 'Van Zyl, Jayd', 'id': 'vzyjoh019', 'email':
'VZYJOH019@myuct.ac.za', 'role': 'Student'}
}
```


STOP 4 – UCT Student IDs

@ 39 BOTHA, CHRISTOPHER BTHCHR020

@ 88 LIE, ANGEL LXXHSI006

@ 89 LIN, CHARLIE LNXCHI026

@ 93 MAHLANZA, THEMBELA MHLTHE035

@ 102 MATHEBULA, MATIMU MTHMAT029

@ 107 MLABA, THABISO MLBTHA016

@ 101 MASHINGAIDZE, STEPHEN MSHKUZ001

@ 110 MOODLEY, SANKESHAN MDLSAN021

@ 120 NAIDOO, KIMESHAN NDXKIM024

@ 121 NAIDOO, NIKAILA NDXNIK006

@ 150 REICH, CHAD RCHCHA006

@ 178 VAN ZYL, JAYD VZYJOH019

@ 186 WESSELS, JANA WSSJAN006

STOP[-1] – Last STOP

- Note
  - **We are assuming STOP is a sequence here—possibly a List**
  - **STOP[-1] == STOP[len(STOP)-1]**
- Part 3 Assessment
  - Test 3
    - Testing, Lists, Dictionaries, Sets& Nested Collections
  - Examination
    - Functions, Testing, Lists, Dictionaries, Sets& Nested Collections
- And...
  - Watch some more Monty Python
    - They uploaded everything on YouTube—in HD