
Data Structures



olsen software

Contents

1. Sequence types
2. Using sequences
3. Set types
4. Mapping types
5. Additional techniques
6. Worked examples



Demo folder: 05-DataStructures

1. Sequence Types

- Overview
- Lists
- Splitting and joining
- Tuples
- Ranges

Overview

- Basic sequence types
 - List, tuple, and range
- Text sequence types
 - String
- Binary sequence types
 - bytes, bytearray, and memoryview

Lists

- There are several ways to create a list

- `[]`
- `[item, item, item ...]`
- `list()`
- `list(iterable)`

- Example:

```
list1 = []
list2 = ["Italy", "France", "Spain"]
list3 = [3, 12, 19, 1, 2, 7]
list4 = list()
list5 = list(list3)
list6 = list("Hello")

print("list1 has %d items: %s" % (len(list1), list1))
print("list2 has %d items: %s" % (len(list2), list2))
print("list3 has %d items: %s" % (len(list3), list3))
print("list4 has %d items: %s" % (len(list4), list4))
print("list5 has %d items: %s" % (len(list5), list5))
print("list6 has %d items: %s" % (len(list6), list6))
```

lists.py

Splitting and Joining

- A common scenario where lists crop up in Python is when you call `split()` or `join()` on a string
 - `split()` – splits a string into a list of substrings
 - `join()` – joins a list into a concatenated string

- Example:

```
str = "and we were singing, hymns and arias, land of my fathers, ar hyd yr nos"  
words = str.split(", ")  
lines = "...\\n".join(words)  
print("%s" % lines)
```

`splitJoin.py`

Tuples

- There are several ways to create a tuple

- `()`
- `a`, or `(a,)`
- `a, b, c` or `(a, b, c)`
- `tuple()`
- `tuple(iterable)`

- Example:

```
tuple1 = ()
tuple2 = "Norway",      # or: tuple2 = ("Norway",)
tuple3 = 3, 19, 2       # or: tuple3 = (3, 19, 2)
tuple4 = tuple()
tuple5 = tuple(tuple3)

print("tuple1 has %d items: %s" % (len(tuple1), tuple1))
print("tuple2 has %d item(s): %s" % (len(tuple2), tuple2))
print("tuple3 has %d items: %s" % (len(tuple3), tuple3))
print("tuple4 has %d items: %s" % (len(tuple4), tuple4))
print("tuple5 has %d items: %s" % (len(tuple5), tuple5))
```

`tuples.py`

Ranges

- To create a range, use the range constructor
 - `range(stop)`
 - `range(start, stop)`
 - `range(start, stop, step)`
- Example:

```
def display_range(msg, r):  
    print("\n" + msg)  
    for i in r:  
        print(i)  
  
range1 = range(5)  
range2 = range(5,10)  
range3 = range(5,10,2)  
  
display_range("range1", range1)  
display_range("range2", range2)  
display_range("range3", range3)
```

`ranges.py`

2. Using Sequences

- Common sequence operations
- Slicing operations
- Unpacking operations
- Sequence modification operations
- Optional exercise

Common Sequence Operations

- You can perform these operations on any sequence:

```
euro = ["GB", "ES", "NL", "F", "D", "I", "P"]
asia = ["SG", "JP"]

print("%s" % "P" in euro)           # True
print("%s" % "F" not in euro)      # False
print("%s" % (euro + asia))        # ['GB', 'ES', 'NL', 'F', 'D', 'I', 'P', 'SG', 'JP']
print("%s" % (asia * 2))            # ['SG', 'JP', 'SG', 'JP']
print("%s" % (2 * asia))            # ['SG', 'JP', 'SG', 'JP']
print("%d" % len(euro))             # 7
print("%s" % min(euro))             # D
print("%s" % max(euro))             # P
print("%d" % euro.index("NL"))      # 2
print("%d" % euro.index("NL", 1))   # 2
print("%d" % euro.index("NL", 1, 4)) # 2
print("%d" % euro.count("ES"))      # 1
```

sequenceCommonOps.py

Slicing Operations

- You can slice the contents of a sequence...

```
euro = ["GB", "ES", "NL", "F", "D", "I", "P"]
asia = ["SG", "JP"]

print("%s" % (euro[1]))           # ES
print("%s" % (euro[1:5]))         # ['ES', 'NL', 'F', 'D']
print("%s" % (euro[1:5:2]))       # ['ES', 'F']
print("%s" % (euro[3:]))          # ['F', 'D', 'I', 'P']
print("%s" % (euro[:3]))          # ['GB', 'ES', 'NL', 'F']
```

slicing.py

Unpacking Operations

- You can unpack (i.e. extract) elements in a sequence
- The following example illustrates the techniques available

```
euro = ["GB", "ES", "NL", "F"]

# Manually getting items.
a, b, c, d = euro[0], euro[1], euro[2], euro[3]
print("%s %s %s %s" % (a, b, c, d))      # GB ES NL F

# Unpacking.
e, f, g, h = euro
print("%s %s %s %s" % (e, f, g, h))      # GB ES NL F

# Catch-all unpacking.
i, j, *k = euro
print("%s %s %s" % (i, j, k))            # GB ES ['NL', 'F']
```

unpacking.py

Sequence Modification Operations

- You can perform these operations on a mutable sequence such as a list:

```
euro = ["GB", "ES", "NL", "F"]

euro[0] = "CY"
euro[1:3] = ["US", "AU", "AT"]
euro.append("SW")
euro.extend(["YU", "ZR"])
euro.insert(1, "NI")
print("%s" % euro)          # ['CY', 'NI', 'US', 'AU', 'AT', 'F', 'SW', 'YU', 'ZR']

euro.pop()
euro.pop(1)
del euro[2:4]
print("%s" % euro)          # ['CY', 'US', 'F', 'SW', 'YU']

euro.remove("US")
euro.reverse()
print("%s" % euro)          # ['YU', 'SW', 'F', 'CY']

eurocopy = euro.copy()
euro.clear()
print("%s" % eurocopy)      # ['YU', 'SW', 'F', 'CY']
print("%s" % euro)          # []
```

sequenceModification.py

Optional Exercise

- Write a Python program as follows:
 - Ask the user to enter a series of numbers (-1 to quit)
 - Determine which numbers are prime
 - Display the prime numbers on the console
- For detailed instructions:
 - See the Notes underneath this slide
- For the solution code:
 - See `C:\PythonDev\Solutions\05-DataStructures\primes.py`

3. Set Types

- Creating a set
- Creating a frozen set
- Common set operations
- Set modification operations

Creating a Set

- There are several ways to create a set

- $\{item, item, item, \dots\}$
- `set()`
- `set(iterable)`
- Via a comprehension, similar to lists

- Example:

```
set1 = {"dog", "ant", "bat", "cat", "dog"}
set2 = set()
set3 = set(("dog", "ant", "bat", "cat", "dog"))
set4 = set("abracadabra")
set5 = {c.upper() for c in "abracadabra"}

print("set1 has %d items: %s" % (len(set1), set1))
print("set2 has %d items: %s" % (len(set2), set2))
print("set3 has %d items: %s" % (len(set3), set3))
print("set4 has %d items: %s" % (len(set4), set4))
print("set5 has %d items: %s" % (len(set5), set5))
```

`sets.py`

Creating a Frozen Set

- Creating a frozenset is similar to creating a set
 - Use the frozenset constructor

- Example:

```
set1 = frozenset({"dog", "ant", "bat", "cat", "dog"})
set2 = frozenset()
set3 = frozenset(("dog", "ant", "bat", "cat", "dog"))
set4 = frozenset("abracadabra")
set5 = frozenset({c.upper() for c in "abracadabra"})
```

```
print("set1 has %d items: %s" % (len(set1), set1))
print("set2 has %d items: %s" % (len(set2), set2))
print("set3 has %d items: %s" % (len(set3), set3))
print("set4 has %d items: %s" % (len(set4), set4))
print("set5 has %d items: %s" % (len(set5), set5))
```

`frozensets.py`

Common Set Operations

- You can perform these operations on any set:

```
s1 = {"GB", "US", "SG"}
s2 = {"GB", "US", "AU"}
s3 = {"F", "BE", "CA"}

print("%s" % ("GB" in s1))          # True
print("%s" % ("GB" not in s1))      # False

print("%s" % (s1.isdisjoint(s2)))   # False
print("%s" % (s1.isdisjoint(s3)))   # True

print("%s" % (s1.issubset(s2)))      # False
print("%s" % (s1 <= s2))             # False
print("%s" % (s1 < s2))              # False

print("%s" % (s1.issuperset(s2)))    # False
print("%s" % (s1 >= s2))             # False
print("%s" % (s1 > s2))              # False

print("%s" % (s1.union(s2, s3)))     # {'GB', 'US', 'BE', 'F', 'CA', 'AU', 'SG'}
print("%s" % (s1 | s2 | s3))         # {'GB', 'US', 'BE', 'F', 'CA', 'AU', 'SG'}

print("%s" % (s1.difference(s2, s3))) # {'SG'}
print("%s" % (s1 - s2 - s3))          # {'SG'}

print("%s" % (s1.symmetric_difference(s2))) # {'AU', 'SG'}
print("%s" % (s1 ^ s2))                # {'AU', 'SG'}
```

setCommonOps.py

Set Modification Operations

- You can perform these operations on a mutable set:

```
s1.add("HK")
s1.remove("US")
s1.discard("D")
print("%s" % s1)          # {'SG', 'HK', 'GB'}
```

```
print("%s" % s1.pop())    # SG
print("%s" % s1)          # {'HK', 'GB'}
```

```
s1.update(s2,s3)
s1 |= s4 | s5
print("%s" % s1)          # {'D', 'AU', 'US', 'I', 'F', 'P', 'N', 'GB', 'CA', 'HK'}
```

```
s1.intersection_update(s2,s3)
s1 &= s4 & s5
print("%s" % s1)          # {'GB', 'US'}
```

```
s1.difference_update({"AA", "BB"}, {"CC", "GB"})
s1 -= {"DD", "EE"} | {"FF", "GG"}
print("%s" % s1)          # {'US'}
```

```
s1.symmetric_difference_update(s2)
s1 ^= s2
print("%s" % s1)          # {'US'}
```



```
s1 = {"GB", "US", "SG"}
s2 = {"GB", "US", "AU"}
s3 = {"GB", "US", "F", "CA"}
s4 = {"GB", "US", "I", "D"}
s5 = {"GB", "US", "P", "N"}
```

setModification.py

4. Mapping Types

- Creating a dictionary
- Iterating over a dictionary
- Accessing items in a dictionary

Creating a Dictionary

- There are several ways to create a dict
 - `{key:value, key:value, ... }`
 - `dict()`
 - `dict(anotherDict)`
 - `dict(keyword=value, keyword=value, ...)`
 - `dict(zip(keysIterable, valuesIterable))`

- Example:

```
dict1 = {"us":"+1", "nl":"+31", "no":"+47"}
dict2 = dict()
dict3 = dict({"us":"+1", "nl":"+31", "no":"+47"})
dict4 = dict(us="+1", nl="+31", no="+47")
dict5 = dict(zip(["us", "nl", "no"], ["+1", "+31", "+47"]))
```

```
print("dict1 has %d items: %s" % (len(dict1), dict1))
print("dict2 has %d items: %s" % (len(dict2), dict2))
print("dict3 has %d items: %s" % (len(dict3), dict3))
print("dict4 has %d items: %s" % (len(dict4), dict4))
print("dict5 has %d items: %s" % (len(dict5), dict5))
```

`dicts.py`

Iterating over a Dictionary

- There are several ways to iterate over a dict
 - Iterate over the items (i.e. key-value pairs)
 - Iterate over the keys
 - Iterate over the values

- Example:

```
dialcodes = {"us": "+1", "nl": "+31", "no": "+47"}
```

```
print("Items:")  
for k,v in dialcodes.items():  
    print(k, v)
```

```
print("\nKeys:")  
for k in dialcodes.keys():  
    print(k)
```

```
print("\nValues:")  
for v in dialcodes.values():  
    print(v)
```

dictIteration.py

Accessing Items in a Dictionary

- There are various operations for accessing items in a dict

```
dialcodes = {"us": "+1", "nl": "+31", "no": "+47", "it": "+39"}

print("%s" % "us" in dialcodes)          # True
print("%s" % "us" not in dialcodes)      # False

dialcodes["uk"] = "+44"
print(dialcodes["uk"])                   # +44
print(dialcodes.get("fr"))               # None
print(dialcodes.get("fr", "xxx"))        # xxx

del dialcodes["no"]
print(dialcodes.pop("uk"))                # +44
print(dialcodes.pop("uk", "xxx"))         # xxx
print(dialcodes.setdefault("it", "???")) # ???

dialcodes.update({"ca": "+1", "it": "+39"})
print(dialcodes)  # {'ca': '+1', 'us': '+1', 'nl': '+31'}
```

dictAccessItems.py

5. Additional Techniques

- Generators
- List comprehensions
- Set comprehensions
- Dictionary comprehensions
- Filtering, sorting, and mapping
- Working with JSON data

Generators

- A generator is a special kind of function that returns a collection, one item at a time
 - Use the `yield` keyword to yield the next value on each call
- Example - consider the following two functions
 - The 1st version returns a collection "all at once"
 - The 2nd version yields a collection one element at a time

```
def getNums():  
    nums = []  
    while True:  
        num = int(input("Number? "))  
        if num == -1 :  
            break  
        nums.append(num)  
    return nums
```

```
# Client code.  
nums = getNums()  
for n in nums:  
    print("  %d" % n)
```

returnCollectionAllAtOnce.py

```
def getNumsB():  
    while True:  
        num = int(input("Number? "))  
        if num == -1 :  
            break  
        yield num
```

```
# Client code.  
nums = getNumsB()  
for n in nums:  
    print("  %d" % n)
```

returnCollectionViaGenerator.py

List Comprehensions

- You can create a list from another sequence
 - Apply an operation on all the items in an existing sequence
 - This is known as a "list comprehension"

- Example:

```
squares = [x**2 for x in range(6)]  
  
ftemps = [32, 68, 212]  
ctemps = [(f-32)*5/9 for f in ftemps]  
  
print("squares: %s" % squares)  
print("ftemps: %s" % ftemps)  
print("ctemps: %s" % ctemps)
```

listComprehensions.py

Set Comprehensions

- You can also create a "set comprehension"
 - i.e. a set created from another sequence

- Example:

```
ftemps = range(0, 50, 5)
ctemps = { int((f-32)*5/9) for f in ftemps }

print("ctemps:  %s" % ctemps)
```

setComprehensions.py

Dictionary Comprehensions

- You can also create a "dictionary comprehension"
 - i.e. a collection of key/value pairs created from another sequence
- Example:

```
mydict = { i : i*i for i in range(1, 6) }
```

```
print("mydict:  %s" % mydict)
```

`dictComprehensions.py`

Filtering, Sorting, and Mapping (1 of 2)

- Python defines functions that allow you to filter, sort, and map (i.e. transform) the elements in a collection

- Example

filteringSortingMapping.py

```
names = ["Zak", "Tim", "Ben", "Joe", "Kim", "Bud", "Ted", "Baz"]
```

```
bnames = list(filter(startswithB, names))  
print(bnames)
```

```
sortedBnames = sorted(bnames)  
print(sortedBnames)
```

```
mappedSortedBnames = list(map(topAndTail, sortedBnames))  
print(mappedSortedBnames)
```

```
def startswithB(element):  
    if len(element) and element[0] == 'B':  
        return True  
    else:  
        return False
```

```
def topAndTail(element):  
    return "****" + element + "****"
```

Filtering, Sorting, and Mapping (2 of 2)

- The `sorted()` function takes two optional arguments, which allow you to take control over the sorting
 - `key` - function that indicates what aspect to sort items on
 - `reverse` - boolean (default is `False`, i.e. ascending order)

- Example

`customSorting.py`

```
names = ["Andy", "Jayne", "Em", "Tom"]

sortedNamesAlphabetically = sorted(names)
print(sortedNamesAlphabetically)

sortedNamesByLength = sorted(names, key=personNameLength)
print(sortedNamesByLength)

sortedNamesByLengthDescending = sorted(names, key=personNameLength, reverse=True)
print(sortedNamesByLengthDescending)
```

```
def personNameLength(p) :
    return len(p)
```

Working with JSON Data (1 of 3)

- JSON is a popular string data format
 - Typically used for passing data to/from REST services
 - Very easy to read/write JSON data in JavaScript (and in Python 😊)

- Here are some example JSON strings:

```
personJson = '{ "name": "Andy", "age": 21, "height": 1.67, "isWelsh": true }'
```

```
coordsJson = '[ { "x": 100, "y": 150 }, { "x": 200, "y": 250 } ]'
```

- To read/write JSON data in Python, use the standard Python module named `json`
 - `json.loads()` loads JSON data into a Python dictionary/list
 - `json.dumps()` dumps a Python dictionary/list into a JSON string

Working with JSON Data (2 of 3)

- These examples show how to load JSON data into Python data structures

```
import json

personJson = '{"name": "Andy", "age": 21, "height": 1.67, "iswelsh": true }'

person = json.loads(personJson)

print("%s is %d years old" % (person["name"], person["age"]))
print("Height is %.2f, welshness is %s" % (person["height"], person["iswelsh"]))
```

jsonLoadObject.py

```
import json

coordsJson = '[ { "x": 100, "y": 150 }, { "x": 200, "y": 250 } ]'

coords = json.loads(coordsJson)

print("Point 0 is %d, %d" % (coords[0]["x"], coords[0]["y"]))
print("Point 1 is %d, %d" % (coords[1]["x"], coords[1]["y"]))
```

jsonLoadList.py

- Also see readJsonFromFile.py and sampledata.json

Working with JSON Data (3 of 3)

- These examples show how to dump Python data into a JSON string

```
import json

person = {"name": "Andy", "age": 21, "height": 1.67, "iswelsh": True }

personJson = json.dumps(person, indent=4)

print(personJson)
```

jsonDumpObject.py

```
import json

coords = [ { "x": 100, "y": 150 }, { "x": 200, "y": 250 } ]

coordsJson = json.dumps(coords, indent=4)

print(coordsJson)
```

jsonDumpList.py

6. Worked Examples

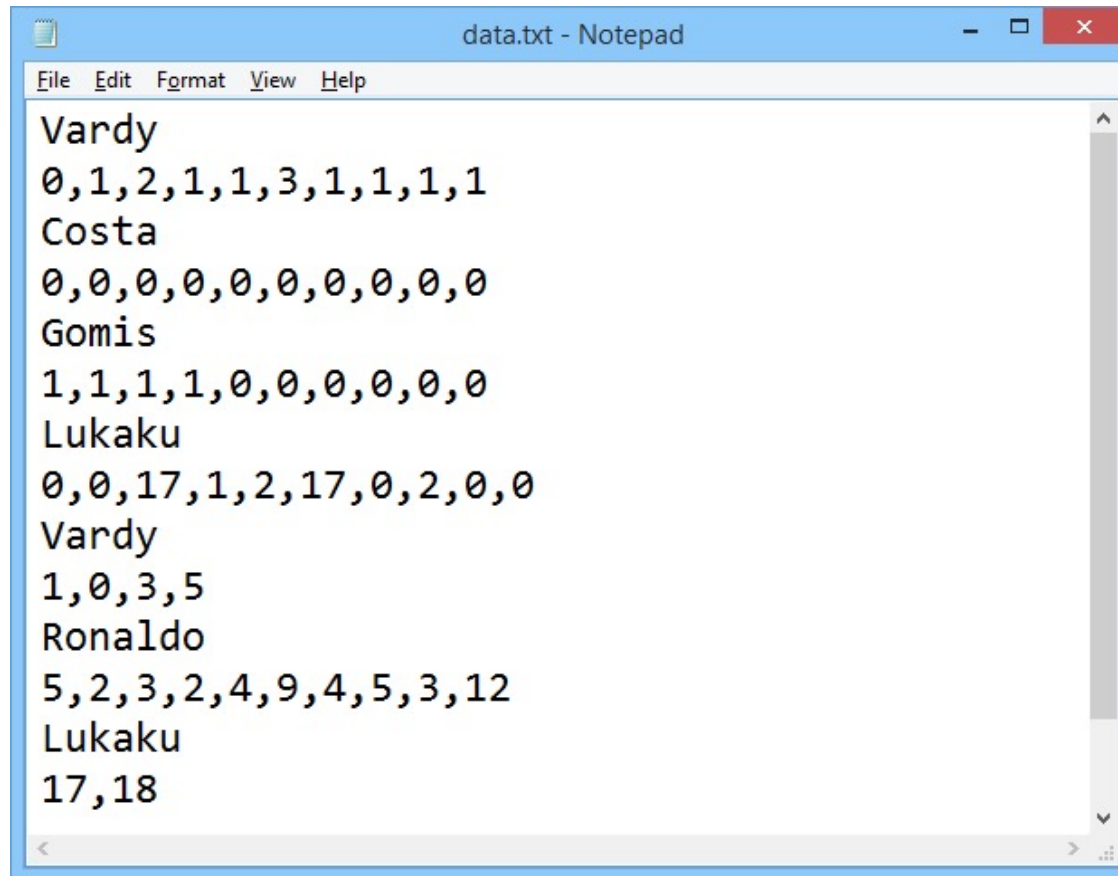
- Example 1
- Example 2

Example 1 - Overview

- We've written an example to illustrate how to use data structures in a real scenario
 - Processing lines of text from a file
- Demo location:
 - `C:\PythonDev\Demos\05-DataStructures\workedExamples`
- Files:
 - `data.txt`
 - `readData.py`

Example 1 - Sample Data

- Here's the sample data for the first application



```
data.txt - Notepad
File Edit Format View Help
Vardy
0,1,2,1,1,3,1,1,1,1
Costa
0,0,0,0,0,0,0,0,0,0
Gomis
1,1,1,1,0,0,0,0,0,0
Lukaku
0,0,17,1,2,17,0,2,0,0
Vardy
1,0,3,5
Ronaldo
5,2,3,2,4,9,4,5,3,12
Lukaku
17,18
```

Example 1 - Application Code

- Here's the code for the first application
 - See the full demo code for detailed comments

```
goals_dict = dict()

# Process data for a player name and the list of goals for that player.
def process_data(name, goals_string):
    goals_list = goals_string.split(',')
    goals_set = set(goals_list)

    if goals_dict.get(name):
        goals_dict[name].update(goals_set)
    else:
        goals_dict[name] = goals_set

# Main code - process the file and populate goals_dict with all the info.
with open('data.txt') as fh:
    for name in fh:
        name = name.rstrip('\r\n')
        goals_string = fh.readline().rstrip('\r\n')
        process_data(name, goals_string)

# Display the goal-scoring pattern for each player.
for k,v in goals_dict.items():
    print("%s\t%s" % (k, v))
```

readData.py

Example 2

- We've written another example that illustrates additional data structures techniques, plus regular expressions
 - Read all the "attribute names" from one file
 - Read all the "attribute names" from another file
 - Determine what "attribute names" are common in both files
- Demo location:
 - C:\PythonDev\Demos\05-DataStructures\workedExamples
- Files:
 - dataFile1.txt
 - dataFile2.txt
 - compareDataFiles.py

Any Questions?

