Topic 6

Collections and Sequences

Murdoch
UNIVERSITY

Last Topic

Murdoch UNIVERSITY

- introduced new **compound data types:** tuple and list
- Access elements of a tuple or list, slicing a tuple and list
  Concatenation of two lists or two tuples
- List comprehension
- idea of mutability
- idea of aliasing and cloning
- command line arguments `sys.argv`

Today

Murdoch UNIVERSITY

- Collections
- Sets
- Dictionaries
- Use of dictionaries to solve problems
- Sequences
- Common characteristics of sequences

Collection

Murdoch UNIVERSITY

- A collection is a group of related values or objects stored together in a single container
  - You can count the number of items in the collection (using the `len` function)
  - You can iterate over its members in a loop
  - You can test whether an item is in a collection (using the `in` and `not in` operator)
  - In some collections, these items are ordered (string, list, and tuple), and in others (set), items have no fixed order
  - Some collections can be changed (mutable, eg, lists). Other collections cannot be changed once they are created (eg, string and tuples).

Collection in Python

Murdoch UNIVERSITY

- Python defined 4 built-in collection types:
  - Tuple: ordered, immutable, can have duplicates
  - List: ordered, mutable, can have duplicates
  - Set: un-ordered, mutable, no duplicates
  - Dictionary: ordered, mutable, no duplicates
- String can also be considered as a collection type.
- In Topic 3 and Topic 5, we have already seen strings, tuples and lists.
- In this topic, we will look at sets and dictionaries.

Sets

Murdoch UNIVERSITY

- Like in mathematics, a set is a collection of unique items.
- In Python, a set is a collection of items. These items do not have a fixed order in the set.
- Each item only appears in a set once, ie, there cannot be more than one occurrence of the same item in a set
- An item in a set cannot be changed, therefore, mutable objects such as lists, sets and dictionaries cannot be members of a set
- However, you can add new items into a set and you can remove items from the set.

Define a Set

Murdoch UNIVERSITY

- A set literal is defined using curly brackets

```
fruit = {"apple","banana","mango","cherry"}
```

- A set may contain values of different types

```
myset = {34,"Python",True,12.8}
```

- It can even have tuples in a set (because a tuple is immutable)

```
s = {"abc",(1,2,3)}
```

- While the following definition would cause an error:

```
s = {"abc",[1,2,3]}
```

Functions for Sets

Murdoch UNIVERSITY

- Most functions applicable to tuples and lists are also applicable to sets

```
print(len(myset)) # => 4

print(type(myset)) # => <class 'set'>

print(isinstance(myset, set)) # => True

print(sorted(fruit)) # return a list

# => ['apple','banana','cherry','mango']

marks = {76, 90, 51, 82}

print(min(marks)) # => 51

print(max(marks)) # => 90

print(sum(marks)) # => 299
```

Indexing for Sets

Murdoch UNIVERSITY

- As the items in a set do not have fixed order, there is no index for an individual item of the set.
- Therefore, we cannot use index to access an item in a set.
- To see that the items in a set have no fixed order, create a small program `set_order.py` (as below) and run it several times. You will see the order of items are not always same at different runs:

```
# set_order.py
s = { "apple", "banana", "cherry"}
print(s)
```

Indexing for Sets

Murdoch UNIVERSITY

```
$ python setOrder.py
{'banana', 'cherry', 'apple'}
$ python setOrder.py
{'cherry', 'banana', 'apple'}
$ python setOrder.py
{'banana', 'apple', 'cherry'}
$
```

Add and Remove Items from a Set

Murdoch UNIVERSITY

- Set has `add` and `remove` methods:

```
s = {1, 2, 3}
s.add(4)              # => {1,2,3,4}
s.remove(3)           # => {1,2,4}
```

- Example: divide Perth post codes into South of River and North of River

```
def perthPostCode(s):
    northSet=southSet = set()    # empty set, cannot use {}
    for x in s:
        if x >= 6100:
            southSet.add(x)
        else:
            northSet.add(x)
    return northSet, southSet

perth = {6008, 6170, 6009, 6051, 6150, 6109}
north, south = perthPostCode(perth)
print("north of river: ", north)    # {6008, 6009, 6051}
print("south of river: ", south)    # {6170, 6150, 6109}
```

Set Union and Intersection

Murdoch UNIVERSITY

- The union of two sets is a new set containing members of both sets

```
s1 = { 1, 2, 3, 'a', 'b'}
s2 = { 'a', 'b', 'c', 1, 2 }
s3 = s1 | s2.  # s3 contains elements of s1 and s2
print(s3)   # {1,2,3,'a','b','c'}
```

Same as

```
s3 = s1.union(s2)
```

- The intersection of two sets is a new set containing members common to both sets

```
s1 = { 1, 2, 3, 'a', 'b'}
s2 = { 'a', 'b', 'c', 1, 2 }
s3 = s1 & s2  # s3 contains elements common to both s1 and s2
print(s3)   # {1,2,'a','b'}
```

Same as

```
s3 = s1.intersection(s2)
```

Dictionary

Murdoch UNIVERSITY

- A dictionary in Python is a list of *key*:*value* pairs. It is an extremely useful data structure. It attaches a unique key to each value, allowing us to access a value by its key, making the program a lot more readable.
- A dictionary is also known as an associative array (PHP), map (C++ and Java), hash (Perl), table (Lua), object (JavaScript).
- In Python, a dictionary is similar to a list: it is mutable and the items in a dictionary is ordered (since Python 3.7). However, there cannot be duplicates in a dictionary because no two elements have the same key).

Define a Dictionary

Murdoch UNIVERSITY

- A dictionary literal is defined using curly brackets, similar to sets. However, each element must be a *key*:*value* pair.

```
student = {
     "name" : "John",
     "major" : "CS",
     "age" : 25
     "mobile" : "0410342765"
}
```

- Since curly brackets are used to define both sets and dictionaries, the following definition can be confusing:

```
x = {}
```

  - as it could be seen as either an empty set or empty dictionary.
  - In Python, the above definition is reserved for empty dictionary, not empty set. To define am empty set, use set constructor instead:

```
x = set()
```

List vs Dictionary

Murdoch UNIVERSITY

A list

0	"John"
1	"CS"
2	25
3	"0410342765"
...	...

A dictionary

"name"	"John"
"major"	"CS"
"age"	25
"mobile"	"0410342765"
…	…

- List: use the index to access a list element
  - The index must be an integer starting from 0. The index only tells you the position of the value in the list, not the nature of the value (eg, whether it is a student's name or his age)
- Dictionary: use the key to access a dictionary element
  - The key reflects the nature of the value, eg, using the key `name` to access the student's name and use the key `age` to access the student's age.

Access Values in a Dictionary

Murdoch UNIVERSITY

- Use the key to access the value

```
print("Student name: " + student['name'])
print("Student age: " + student['age'])
```

- As a dictionary is an object, it has many built-in methods. Another way to access the value is by using the `get` method

```
print("Student name: " + student.get('name'))
print("Student age: " + student.get('age'))
```

- Obtain all keys

```
print(student.keys()) # ["name", "major", "age", "mobile"]
```

- Obtain all values

```
print(student.values()) # ["John", "CS", 25, "0410342765"]
```

Add and Remove Items from a Dictionary

Murdoch UNIVERSITY

- To add a new item (a key:value pair), use a new key

```
student['gender'] = 'Male' # 'gender" is the key
student['address'] = '20 South St, Murdoch'
```

- Alternatively, one may use `update` method to add one or more new items to a dictionary

```
x = {'gender':'Male', 'address':'20 South St, Murdoch'}
student.update(x)
```

- To remove an item from the dictionary and return it

```
x = student.pop['address']
print('address is %s" %x) # '20 South St, Murdoch'
```

- To pop off the last item from the dictionary

```
x = student.popitem() # ('gender', 'Male')
```

Delete Items from a Collection

Murdoch UNIVERSITY

- To delete an item with a given key from the dictionary

```
del student['gender']
```

- `del` is not a function or method. It is a statement. It can be used to delete an item from a list as well

```
student = [ 'John', 'CS', 25 ]
del student[1]   # remove 'CS' from the list
```

- `del` can be used to delete an entire collection.

```
a = { 1, 2, 3 }
b = [ "abc", "xy" ]
c = ( 10.5, "foo", True )
d = { 'a' : 1, 'b' : 2 }
del a, b, c, d
```

Delete Unneeded Objects

Murdoch UNIVERSITY

- An object such as a string, a list, a dictionary, etc, can take up a lot of system resources (memory space and computer processing time). When an object is no longer needed, you should delete it to save the system resources.

```
x = 10
y = "Hello, world!"
del x, y
```

- Once deleted, an object is completely removed from your process (running program). Any attempt to use it will generate an error.
- However, you can always define a new variable using the variable name that has been deleted.

Loop through a Dictionary

Murdoch UNIVERSITY

- You can loop through a dictionary using its keys, values, and both keys and values

```
grades = { 'HD':4, 'D':10, 'C':23, 'P':34, 'N':10 }
numStudents = 0
for x in grades.values():
    numStudents += x
print("Total number of students is ", numStudents)

# print the grade distributions
for x in grades.keys():
   print("%s : %0.1f  " %(x, grades[x]/numStudents * 100))

# alternatively
for k,v in grades.items():
   print("%s : %0.1f  " %(k, v/numStudents * 100))
```

Dictionary: Alias and Clone

Murdoch UNIVERSITY

- In Topic 5, we discussed object aliasing and cloning
  - An alias is like to give a nickname to a variable, both the original variable and the new variable are in fact representing the same object.
  - While a clone of an object is a separate copy of the original object. The original variable and the new variable represent two different objects, albeit the new objects have the same initial value as the original object.

```
john = { 'name':'John', 'age':25 }
david = john   # david is an alias of john
david['name'] = 'David' # john's name is also changed to David.
```

- To create a clone of a dictionary, use `copy` method

```
peter = john.copy() # peter is a clone of john
peter['name'] = 'Peter' # this change would not affect john
```

Sequence Types

Murdoch UNIVERSITY

- In Python, a sequence is a type of collection that holds an ordered sequence of values. Each value is identified by an index.
- We have already seen some of the sequence types
  - String, such as `"Python"`
  - Tuple, such as `("Apple", "Cherry", "Gooseberry")`
  - List, such as `["abc", 1, 3.1415, True]`
  - Range, is a sequence of numbers, such as
    `range(1,10,2)` => 1,3,5,7,9

Common Characteristics of Sequence Types

Murdoch UNIVERSITY

- Sequence types share many common operations and characteristics (with exceptions)
  - Indexing
  - Slicing
  - Repetition, eg, `"abc"*3` => "abcabcabc"
  - Length: such as `len([1,2,3,4,5])` => 5
  - Membership: `in` operation and `not in` operator
  - Iterable: as in the for loop
- Note, some of sequences are mutable, such as lists, others are immutable, such as strings, tuples and ranges

Sequence: Indexing

Murdoch UNIVERSITY

- Each member of a sequence is accessible using its index. The index always starts from 0.
  - String

```
s = "Python is fun!"
s[0]                # 'P'
s.index("fun")      # 10
```

  - Tuple

```
t = ("Apple", "Cherry", "Gooseberry")
t[2]                #  "Gooseberry"
t.index("Cherry")   # 1
```

  - List

```
l = [1, "ICT582", 3.14, True]
l[1]                 #  "ICT582"
l.index(3.14)        #  2
```

Sequence: Indexing

Murdoch UNIVERSITY

- Range

```
r = range(10)   # 0,1,2,3,4,5,6,7,8,9
r[2]            # 2
r.index(4)      # 4, ie, value 4's index is 4

r2 = range(1,10,2) # 1,3,5,7,9
r2[2]           #  5
r2.index(5)     #  2, ie, value 5's index is 2
```

Sequence: Slicing

Murdoch UNIVERSITY

- The same sequence slicing operations apply to all sequence types.
  - String

```
s = "Python is fun!"
fun = s[10:]                    # 'fun!'
```

  - Tuple

```
t = ("Apple", "Cherry", "Gooseberry")
t[::2]                  #  ("Apple", "Gooseberry")
```

  - List

```
l = [1, "ICT582", 3.14, True]
l[1::2]                   #  ["ICT582", True]
```

  - Range

```
r = range(10)   # 0,1,2,3,4,5,6,7,8,9
list(r[::-1])   # [9,8,7,6,5,4,3,2,1,0]
```

Sequence: Concatenation

Murdoch UNIVERSITY

- The same concatenation operations apply to all sequence types (except ranges).
  - String

```
s = "Python"
s + " is fun!"          # 'Python is fun!'
```

  - Tuple

```
t1 = ("Apple", "Cherry")
t2 = ("Orange", "Mango")
t1 + t2   #("Apple","Cherry","Orange","Mango")
```

  - List

```
l1 = ["ICT582", "ICT580"
l2 = ["ICT375", ICT286"]
l1 + l2 # ["ICT582","ICT580","ICT375",ICT286"]
```

  - Range

Concatenation operation doesn't apply to ranges

Sequence: Repetition

Murdoch UNIVERSITY

- A sequence, except ranges, can be repeated many times using operator *
  - String

```
s = "Python!"
s*3          # '"Python!Python!Python!"
```

  - Tuple

```
t = (1,2,3)
t*3          #  (1,2,3,1,2,3,1,2,3)
```

  - List

```
l = ['a','b','c']
l*2             #  ['a','b','c','a','b','c']
```

  - Range

Repetition operation doesn't apply to ranges

Sequence: Length

Murdoch UNIVERSITY

- The length function `len` applies to all sequences
  - String

```
s = "Python!"
len(s)          # 7
```

  - Tuple

```
t = (1,2,3)
len(t)          #  3
```

  - List

```
l = ['a','b','c']
len(l)          #  3
```

  - Range

```
r = range(1,10,2) # 1,3,5,7,9
len(r)          #  5
```

Sequence: Membership

Murdoch UNIVERSITY

- Use membership operators `in` and `not in` to determine whether a value is in the sequence or not
  - String

```
s = "Python!"
'P' in s          # True
"on" in s         # True
```

  - Tuple

```
t = (1,2,3)
2 not in t        #  False
```

  - List

```
l = ['a','b','c']
'x' in l           #  False
```

  - Range

```
r = range(1,10,2) # 1,3,5,7,9
3 in r             #  True
```

Sequence: Iterable

Murdoch UNIVERSITY

- You can iterate over the elements in any sequence in a for loop

```
for element in sequence:
    statement
    . . . . . .
    statement
```

Example: Word Frequency

Murdoch UNIVERSITY

```
def  lyrics_to_frequencies(words):
    """
    input: words is a list of words in a lyrics
    output: return a dictionary containing the
            frequency of each word
    """
    myDict = {}
    for word in words:
        if word in myDict:
            myDict[word] += 1
        else:
            myDict[word] = 1
        return myDict
```

Example: Word Frequency

Murdoch UNIVERSITY

```
def  most_common_words(freqs):
     """
     input: freqs is a dictionary containing
          the frequncies of words in a lyrics
     output: return a tuple containg the list list of
           words with the highest frequency
     """
     values = freqs.values()
     best = max(values)
     words = []
     for k in   freqs:
          if  freqs[k] == best:
               words.append(k)
     return (words, best)
```

Example: Word Frequency

Murdoch UNIVERSITY

```
def words_often(freqs, minTimes):
    """
    input: freqs is the dictionary containing the
       frequencies of words, minTimes is th
          minmum frequency required
    output: return those words with the minimum
          frequencies or higher
    """
    result = []
    done = False
    while not done:
        temp = most_common_words(freqs)
        if temp[1] >= minTimes:
            result.append(temp)
            for w in temp[0]:
                del(freqs[w])
        else:
            done = True
    return result
```

Example: Word Frequency

```
beatles = '''
            Love, love me do
          You know I love you
          I'll always be true
        So please, love me do
          Whoa, love me do

            Love, love me do
          You know I love you
          I'll always be true
        So please, love me do
          Whoa, love me do

           Someone to love
             Somebody new
           Someone to love
           Someone like you
```

```
            Love, love me do
          You know I love you
          I'll always be true
        So please, love me do
          Whoa, love me do

            Love, love me do
          You know I love you
          I'll always be true
        So please, love me do
          Whoa, love me do
            Yeah, love me do
         Whoa, oh, love me do
'''
```

```
wordList = beatles.split()
wordFreq = lyrics_to_frequencies(wordList)
print(words_often(wordFreq, 5))
```

Example: Word Frequency

Murdoch UNIVERSITY

```
hong$ py word_freq.py
[(['love'], 20), (['me', 'do'], 14), (['you', 'Whoa,'], 5)]
hong$
```

References

Murdoch UNIVERSITY

- Python data structures: https://docs.python.org/3/tutorial/datastructures.html
- The Python Tutorial: https://docs.python.org/3/tutorial/
- W3School Python Tutorial https://www.w3schools.com/python/
- Python 3 documentation https://docs.python.org/3/