Applied Computing with Python

SAT 4650

Lecture 4

Dr. Neerav Kaushal

Department of Applied Computing

Outline

- Dictionary
- Set

Set

Sets

- A set is an unordered collection of items where:
  - every element is unique
  - every element can be iterable and immutable
- However, the set itself is mutable. There are no duplicate elements in a set and you can add or even delete items in it
- Sets can perform mathematical operations like intersection, union, symmetric difference, and so on
- A set can consist of elements or items with different immutable data types like integer, string, float, tuple, etc.
- A set cannot have mutable elements like lists, dictionaries or a set itself
- Defined by curly braces – {}
- Duplicates get eliminated
- Sets do not support indexing

```
In [86]: my_set = {1,2,3,4,5}

In [87]: my_set2 = {1,2,3,3,4,4}

In [88]: my_set2
Out[88]: {1, 2, 3, 4}
```

```
In [89]: {1, 2, 3, [4,5,6]}
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[89], line 1
----> 1 {1, 2, 3, [4,5,6]}

TypeError: unhashable type: 'list'
```

```
In [90]: {1, 2, 3, (4,5,6)}
Out[90]: {(4, 5, 6), 1, 2, 3}
```

```
In [91]: a = {1,2,3,4,5}

In [92]: b = {4,5,99,98,97}

In [93]: a-b
Out[93]: {1, 2, 3}
```

```
In [95]: b-a
Out[95]: {97, 98, 99}
```

```
In [97]: a
Out[97]: {1, 2, 3, 4, 5}

In [98]: a[4]
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[98], line 1
----> 1 a[4]

TypeError: 'set' object is not subscriptable
```

Michigan Tech

Sets - 2

- You can initialize an empty set by using set()
- To initialize a set with values, you can pass a list to set()
- You can use the add method to add a new element and remove to remove an existing element from a set
- discard can also delete an element from a set, but it does not throw an error if the element does not exist in the set unlike remove
- pop removes and returns an arbitrary element from a set
- Use clear to empty the list

```
In [100]: empty_set = {}

In [101]: empty_set
Out[101]: {}
```

```
In [102]: students = set( ['claire', 'meera', 'jones', 'kaitlyn', 'carter', 'tom', 'adam'] )

In [103]: students
Out[103]: {'adam', 'carter', 'claire', 'jones', 'kaitlyn', 'meera', 'tom'}
```

```
In [104]: students.add("APPLE")

In [105]: students
Out[105]: {'APPLE', 'adam', 'carter', 'claire', 'jones', 'kaitlyn', 'meera', 'tom'}
```

```
In [106]: students.remove('claire')

In [107]: students.remove('meera')

In [108]: students
Out[108]: {'APPLE', 'adam', 'carter', 'jones', 'kaitlyn', 'tom'}
```

```
In [111]: students
Out[111]: {'APPLE', 'carter', 'jones', 'kaitlyn', 'tom'}

In [112]: students.clear()

In [113]: students
Out[113]: set()
```

Sets - 3

- Like many sequence datatypes in python, you can loop through sets
- You can convert set to list by using sorted() function
- One of the major function of set is to make a list/tuple unique
- Use copy() method to copy a set to a new set
- Use update() to add elements from one list to another to update it
- A common use of sets in Python is computing standard math operations such as union, intersection, difference, and symmetric difference
- Memebership to see if some element(s) are in a set: via in or issubset()

```
In [115]: my_set = {2, 4, 6, 8, 78, 98, 104}

In [116]: for ele in my_set:
     ...:     print(ele)
     ...:
2
98
4
6
8
104
78
```

```
In [127]: odds = {1,3,5,7,9}

In [128]: evens = {2,4,6,8,10}

In [129]: odds.update(evens)

In [130]: odds
Out[130]: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

```
In [117]: my_set
Out[117]: {2, 4, 6, 8, 78, 98, 104}

In [118]: sorted(my_set)
Out[118]: [2, 4, 6, 8, 78, 98, 104]

In [119]: sorted(my_set, reverse=True)
Out[119]: [104, 98, 78, 8, 6, 4, 2]
```

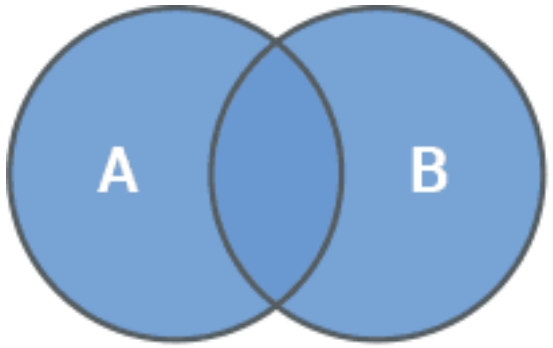
```
In [121]: even_list = [1,2,2,4,6,8,8,12,12,44,44,66]

In [122]: list(set(even_list))
Out[122]: [1, 2, 66, 4, 6, 8, 12, 44]
```

```
In [124]: my_flowers = {'rose', 'lilly', 'sunflower', 'lotus'}

In [125]: your_flowers = my_flowers.copy()

In [126]: your_flowers
Out[126]: {'lilly', 'lotus', 'rose', 'sunflower'}
```

```
In [146]: pc_set = {'hp', 'lenovo', 'dell', 'apple', 'sony', 'google'}

In [147]: 'compaq' in pc_set
Out[147]: False

In [148]: os_set = {'sony', 'google'}

In [149]: os_set.issubset(pc_set)
Out[149]: True
```

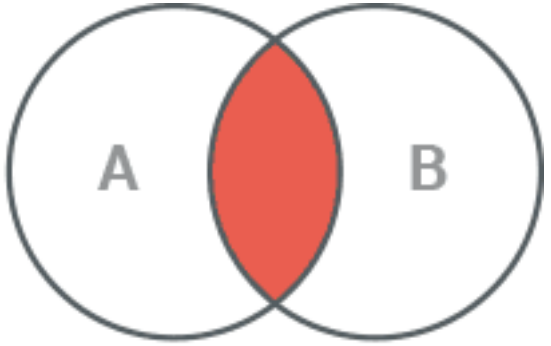
Sets - 4

- Union of 2 sets contains all elements that are either in set A or in set B
- Intersection contains elements that are in both A and B
- Difference between A and B i.e. A-B contains all elements that are in A but not in B
- Symmetric difference contains elements in A or B but not in both

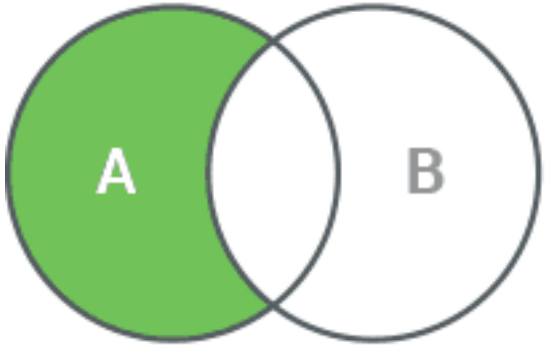



```
a = {1,2,3,4,5}

b = {4,5,6,7,8}
```

```
In [134]: a.union(b)
Out[134]: {1, 2, 3, 4, 5, 6, 7, 8}

In [135]: a | b
Out[135]: {1, 2, 3, 4, 5, 6, 7, 8}
```

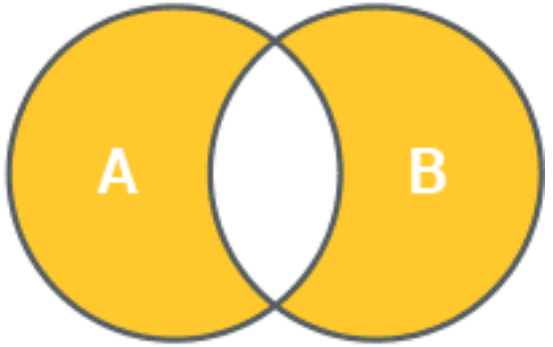



```
In [136]: a.intersection(b)
Out[136]: {4, 5}

In [137]: a & b
Out[137]: {4, 5}
```




```
In [138]: a.difference(b)
Out[138]: {1, 2, 3}

In [139]: a - b
Out[139]: {1, 2, 3}
```

A-B == B-A?




```
In [140]: a.symmetric_difference(b)
Out[140]: {1, 2, 3, 6, 7, 8}

In [141]: a ^ b
Out[141]: {1, 2, 3, 6, 7, 8}
```

Michigan Tech

Sequence Comprehension

List comprehensions

- List comprehensions provide a concise way to create lists
- The list comprehension starts with a '[' and ']', square brackets, to help you remember that the result is going to be a list

```
[i**2 for i in [0,2,4,8,10]]
```
✓ 0.0s
```
[0, 4, 16, 64, 100]
```

```
import math
[round(math.sin(i),2) for i in range(0,6,1)]
```
✓ 0.0s
```
[0.0, 0.84, 0.91, 0.14, -0.76, -0.96]
```

```
my_list = [i for i in range(0, 20, 2)]

my_list
```
✓ 0.0s
```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

```
my_list = []
for i in range(0,20,2):
    my_list.append(i)


my_list
```
✓ 0.0s
```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

```
for item in list:
    if conditional:
        expression
```

```
[expression for item in list if conditional]
```

Michigan Tech

List comprehensions - 2

```
text = "This is a sample line for SAT4650"
print(text, "\n")

listy = text.split(" ")
print(listy, "\n")

lets = [word[0] for word in listy]
print(lets, '\n')

ups = [word.upper() for word in listy]
print(ups)
```

[10] ✓ 0.0s

```
This is a sample line for SAT4650

['This', 'is', 'a', 'sample', 'line', 'for', 'SAT4650']

['T', 'i', 'a', 's', 'l', 'f', 'S']

['THIS', 'IS', 'A', 'SAMPLE', 'LINE', 'FOR', 'SAT4650']
```

```
x = [10, 20, 30]
y = [1, 2, 3]

z = [i+j for i in x for j in y]
print(z)
```

[12] ✓ 0.0s

```
[11, 12, 13, 21, 22, 23, 31, 32, 33]
```

List comprehensions - 3

```
x = list(range(100,200,5))
print('x:', x, '\n')

y = []
for idx in range(len(x)):
    if x[idx]%10 == 0 and x[idx]%3 == 0:
        y.append(x[idx])

print('y:', y)
```

[22] ✓ 0.0s

```
x: [100, 105, 110, 115, 120, 125, 130, 135, 140, 145, 150, 155, 160, 165, 170, 175, 180, 185, 190, 195]

y: [120, 150, 180]
```

```
y = [x[idx] for idx in range(len(x)) if x[idx]%10==0 and x[idx]%3==0]
print('y:', y)
```

[23] ✓ 0.0s

```
y: [120, 150, 180]
```

Michigan Tech

Dictionary comprehensions

- Just like list comprehensions, we can create dict via comprehension
- The dict comprehension starts with a '{' and '}' and colons between key and values to help you remember that the result is going to be a dictionary

```
my_dict = {}

for i in range(1,6,1):
    my_dict[i] = i**2

print(my_dict)
```

[24] ✓ 0.0s

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

```
my_dict = {i : i**2 for i in range(1,6,1)}
my_dict
```

[25] ✓ 0.0s

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

```
{i.upper() : i*3 for i in "python"}
```

[28] ✓ 0.0s

```
{'P': 'ppp', 'Y': 'yyy', 'T': 'ttt', 'H': 'hhh', 'O': 'ooo', 'N': 'nnn'}
```

Michigan Tech

Chaining

- Method chaining means calling multiple methods one after another on the same object in a single expression, where each method returns a new object that the next method operates on
- In python strings, methods like upper() and replace() do not modify the original string; instead, each call returns a new string, which makes chaining possible
- Chaining improves readability and conciseness by expressing a sequence of transformations step-by-step without creating intermediate variables

```
text = "   hEllo wORld!!!      "

text.strip().lower().replace('!', '').title()
```

✓ 0.0s

```
'Hello World'
```

```
a = "   apple pie  "

result = (
    a.strip()
     .upper()
     .replace(" ", "_")
     .startswith("APPLE")
)

print(result)
```

✓ 0.0s

```
True
```

Michigan Tech