COMP10001

2023 Sem 1

Foundations of Computing

Week 6

Tutorial/Workshop ◦ ◦ ◦ ◦

Today's Tutorial

1. Dictionaries and sets
2. None type
3. "in-place" changes

Exercises 1&2

Exercise 1 answer

```
print("We need some saws")
print("We need some hammers")
print("We need some cogs")
print("We need some nails")
```

Very repetitive

```
def get_str(part):
    return f"We need some {part}"

print(get_str("saws"))
print(get_str("hammers"))
print(get_str("cogs"))
print(get_str("nails"))
```

Repetitive
Okay for small input

```
def get_str(part):
    return f"We need some {part}"

parts = ("saws", "hammers", "cogs", "nails")

for part in parts:
    print(get_str(part))
```

No repetition
Very good for larger input

} Will both become 100+ lines for 100 tools

Exercise 2 answer

2. Consider the following `while` loop and two conversions to `for` loops. Are the two `for` loops equivalent? Why might you choose one over the other?

```
count = 0
items = ('eggs', 'spam', 'more eggs')
while count < len(items):
    print(f"need to buy {items[count]}")
    count += 1
```

```
items = ('eggs', 'spam', 'more eggs')
for count in range(len(items)):
  print(f"need to buy {items[count]}")
```

```
items = ('eggs', 'spam', 'more eggs')
for item in items:
  print(f"need to buy {item}")
```

Both **for** loops are equivalent

Second loop is cleaner

Dictionaries

Stores items in **key:value** pairs

Keys and value can be any data type, and can vary item to item

Defined using curly brackets or using dict()

main.py +

```
1
2  dict_1 = {"name": "Catie", "age": 21}
3
4  dict_2 = dict([("name", "Catie"), ("age",21)])
5
6  print(dict_1)
7  print(dict_2)
```

```
{'name': 'Catie', 'age': 21}
{'name': 'Catie', 'age': 21}
```

Dictionary methods

```
my_dict = {"name": "Catie", "age": 21}

my_dict["year"] = 2001
  my_dict = {"name": "Catie", "age": 21, "year": 2001}

my_dict["age"] = 2001
 my_dict = {"name": "Catie", "age": 2001}

my_dict.pop("age")
 my_dict = {"name": "Catie"}
```

key is required for pop

Dictionary methods

```
my_dict = {"name": "Catie", "age": 21}

my_dict.keys()
  dict_keys(['name', 'age'])

my_dict.values()
  dict_values(['Catie', 21])

my_dict.items()
  dict_items([('name', 'Catie'), ('age', 21)])
```

Sets

A **data type**

(one of 4 collections, dict / tuple / list / set)

Has no **duplications**
Has no **order** (hence cannot index it, unlike list)

e.g. my_set = {cat, dog, horse}

Sets

A **data type**

(one of 4 collections, dict / tuple / list / set)

Has no **duplications**

Has no **order** (hence cannot index it, unlike list)

e.g. my_set = {cat, dog, horse}

Set operations

set_a = {"cat", "dog", "horse"}

set_b = {"snake", "dog", "rabbit"}

Union

set_a | set_b = {"cat", "dog", "horse", "snake", "rabbit"}

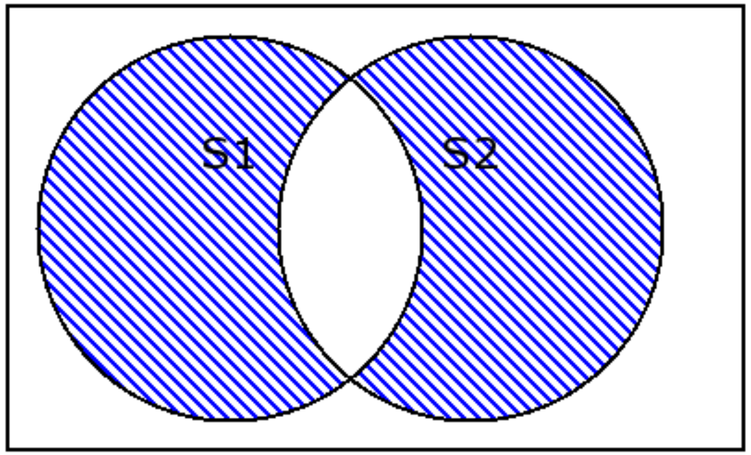
Intersection

set_a & set_b = {"dog"}

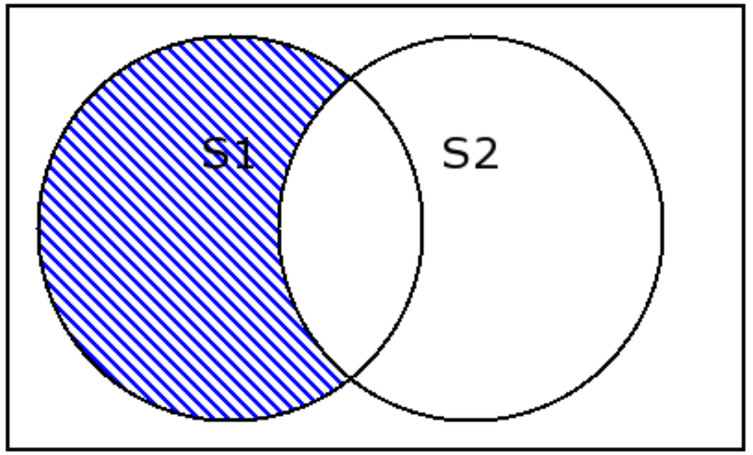
Difference

set_a – set_b = {"cat", "horse"}

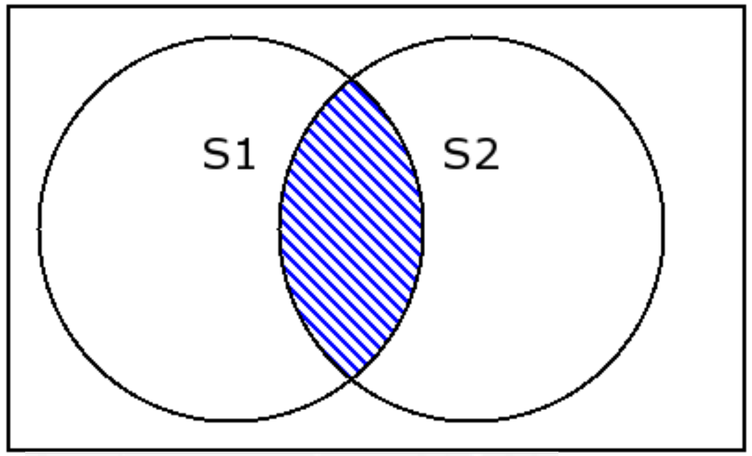
set_b – set_a = {"snake", "rabbit"}



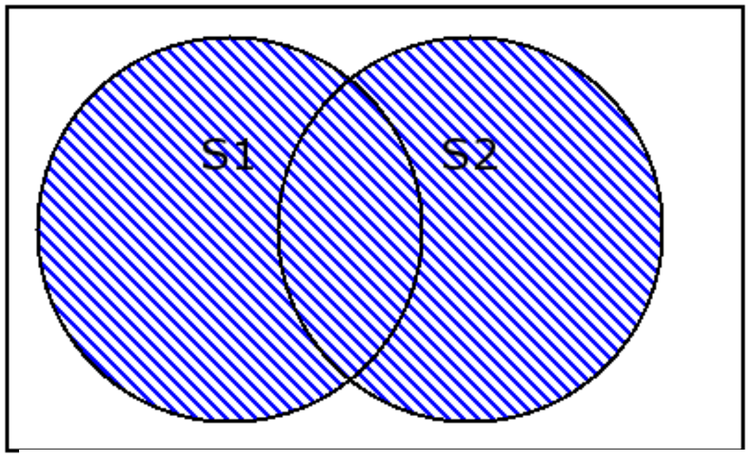
Set Symmetric Difference, S2^S2



Set Difference, S1-S2



Set Intersection, S1&S2



Exercises 3&4

Exercise 3 answer

3. Evaluate the following given the assignment `d = {"R": 0, "G": 255, "B": 0, "other": {"opacity": 0.6}}`. Specify whether the value of `d` changes as a result. Assume `d` is reset to its original value each time.

(a) `"R"in d`

A: *`True` (test for membership among keys)*

(b) `d["R"]`

A: *`0`*

(c) `d["R"] = 255`

A: *New value of `d`:*
`{'R': 255, 'G': 255, 'B': 0, 'other': {'opacity': 0.6}}`

(d) `d["A"]`

A: *`KeyError: 'A'` (requesting a nonexistent key gives an error)*

(e) `d["A"] = 50`

A: *New value of `d`:*
`{'R': 0, 'G': 255, 'B': 0, 'other': {'opacity': 0.6}, 'A': 50}`
(assigning to a nonexistent key adds that (key: value) pair to the dictionary)

Exercise 3 answer

(f) `d.pop("G")`

A: *`255` (removes key from dictionary, returning its value)*
New value of `d`:
`{'R': 0, 'B': 0, 'other': {'opacity': 0.6}}`

(g) `d["other"]["blur"] = 0.1`

A: *New value of `d`:*
`{'R': 0, 'G': 255, 'B': 0, 'other': {'opacity': 0.6, 'blur': 0.1}}`

(h) `d.items()`

A: *`dict_items([('R', 0), ('G', 255), ('B', 0), ('other', {'opacity': 0.6})])`*

Exercise 4 answer

4. Evaluate the following given the assignment `s1 = {1, 2, 4}` and `s2 = {3, 4, 5}`. If `s1` or `s2` change as a result, give their new value. Assume `s1` and `s2` are reset to their original values each time.

(a) `s1.add(7)`

A: *New value of `s1`:*
`{1, 2, 4, 7}`

(b) `s1.add(2)`

A: *`s1` does not change (2 is already in the set)*

(c) `s2.remove(5)`

A: *New value of `s2`:*
`{3, 4}`

(d) `s1 & s2`

A: *`{4}`*

(e) `s1.union(s2)`

A: *`{1, 2, 3, 4, 5}`*

(f) `s1 - s2`

A: *`{1, 2}`*

None type

A **value** in Python

Is not the same as False,

or 0, or empty string

Is essentially **nothing**

```
x = None

if x:
  print("Do you think None is True?")
elif x is False:
  print ("Do you think None is False?")
else:
  print("None is not True, or False, None is just None...")
```

```
None is not True, or False, None is just None...
```

Sorted()

- sorts a list
- returns new list
- does not change the list

```
e.g. L = [2,1]
sorted(L)
print(L)
> [2, 1]
```

.sort()

- sorts a list
- returns nothing (None)
- mutates the list

```
e.g. L = [2,1]
L.sort()
print(L)
> [1, 2]
```

Sorted()

- sorts a list
- returns new list
- does not change the list

```
e.g. L = [2,1]
sorted(L)
print(L)
> [2, 1]
```

not in-place

.sort()

- sorts a list
- returns nothing (None)
- mutates the list

```
e.g. L = [2,1]
L.sort()
print(L)
> [1, 2]
```

in-place

In-place

Editing an object "in-place" means to **mutate** (change) the object itself

Can be good; efficient

Can also be problematic; cannot get the unchanged object back

WORKSHOP

Grok, problems from sheet, ask me questions :)

13

THURSDAY, 13 APRIL 2023 AT 12:00 UTC+10

Codebrew 2023

Arts West, University of Melbourne

See you next week!