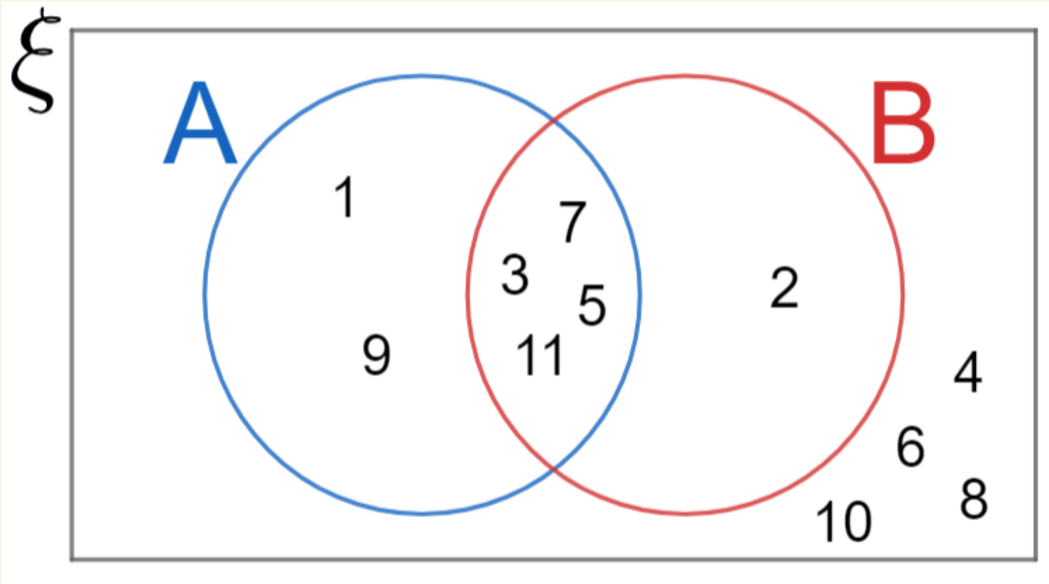
Data Structures

Univ.AI

Sets in mathematics



(from https://mathsmadeeasy.co.uk/gcse-maths-revision/venn-diagrams/)

Various sets can be seen in this diagram, which are useful to know about.

(1) : , set of all odd numbers OR prime numbers

(2) : , set of all odd numbers AND prime numbers

(3) : , the complement of , the set of evens in the "universe"

(4) , the set of all elements in that are not in . Its

: set of numbers from 1 to 11 inclusive,
is called the **universal set** or **universe**.

: set of odd numbers in the "universe"
: set of prime numbers in the "universe"

Sets: collection of unique, immutable objects

Sets can only contain **immutable**, that is unchangeable objects, like ints, floats, strings. Not lists.

Python Code	What it does
`myset = set()`	Create an empty set
`myset.add(element)`, `myset.discard(element)`	Add or remove elements.
`x in myset`	is x in the set? (works for lists as well)
`myset.issubset(anotherset)`, `set <= anotherset`	is `myset` a subset of `anotherset`?
`myset < anotherset`	is `myset` a proper subset of `anotherset`?
`myset.union(set1, ste2,..)`, also written `myset.union(*sets)`	The union of `myset` with one or more other sets, written as arguments to `union`.
`myset.intersection(*sets)`	The intersection of `myset` with one or more other sets
`myset.difference(*sets)` , `myset - anotherset`	elements in `myset` that are not there in the others
`myset.symmetric_difference(anotherset)`	elements in either `myset` or `anotherset` but not in both
`myset.extend(*sets)`	extend `myset` with elements from other sets or lists

Read a file, parse lines, and get all unique words

```
wordset = set() # make a set with unique items
fd = open("filename")
lines = fd.readlines()
fd.close()
# strip newline characters and other whitespace off the edges
cleaned_lines = [line.strip() for line in lines]
# make a list of lists.
# each inner list if the list of words on that line
list_of_lines_words = [line.split() for line in lines]
# Take each list of words, and get all the unique words
for lines_words in list_of_lines_words:
  wordset.update(lines_words) # update the wordset using the new list.
  # duplicates will be removed
# Use list constructot to make a list from the set
unique_words = list(wordset)
```

Dictionaries: look up a value by a key

Python Code	What it does
`d = dict(name='Alice', age=18)`, `d = {'name' : 'Alice', 'age' : 18 }`	Create a dictionary using the `dict` constructor or the "literal" braces notation
`d['name']`, `d.get('name', 'defaultname')`	Access value at key. Second form returns a default if `name` is not in `d`
`d2 = dict(gender='F')`, `d1.update(d2)`	Update from another dictionary
`d['gender'] = 'F'`	Set a value associated with a key
`d.setdefault('gender', 'F')`	If there is a value associated with gender, return it, else **set** that value to default `F` and return it.
`del d['gender']`	delete a key-value pair from the dictionary.
`'gender' in d`	Returns true if the key `gender` is in the dictionary
`d.keys()`	Returns a view over the keys in the dictionary that can be iterated or looped over
`d.values()`	Returns a view over the values in the dictionary which can be iterated or looped over
`d.items()`	Returns a view over pairs (tuples) of type key, value which can be iterated or looped over

Create a dictionary:

```
d = dict(
    name = 'Alice',
    age  = 18,
    gender = 'F'
)
```

Get a value:

```
print("age", d['age'])
```

```
age 18
```

Set a value:

```
d['job] = 'scientist'
```

Iterate over keys:

```
for key in d.keys():
    print(key, d['key'])
```

```
name Alice
age 18
gender F
```

Iterate over keys and values:

```
for key, value in d.items():
    print(key, value)
```

```
name Alice
age 18
gender F
```

Tuples and immutability

Tuples are like lists but they cannot be changed in place. They are often used for storing *different kinds* of data, while lists are used for the same kind. They are **immutable**. Why use them? They are fast! More about this later.

```
tup = ('Alice', 18, 'F')
```

This looks similar to our dictionary, but you cant add any entries to it. Its a *final object*, so to speak.

Why immutability?

- Strings are immutable too!
- Immutability makes things faster.
- Immutable objects can be used as keys in dictionaries!

```
tup = (1, 2, 3)
tup[1] = 4 # replace 2 by 4
```

```
TypeError: 'tuple' object does not support item assignment
```

```
mystr = "Hello World"
mystr[2] = 'k' # replace l with k
```

```
TypeError: 'str' object does not support item assignment
```