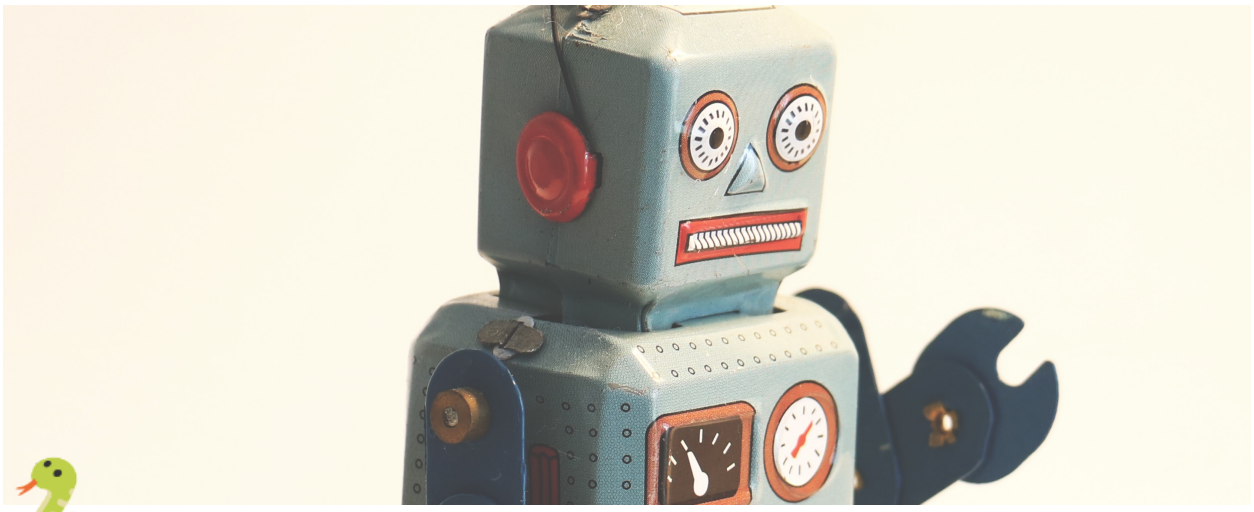
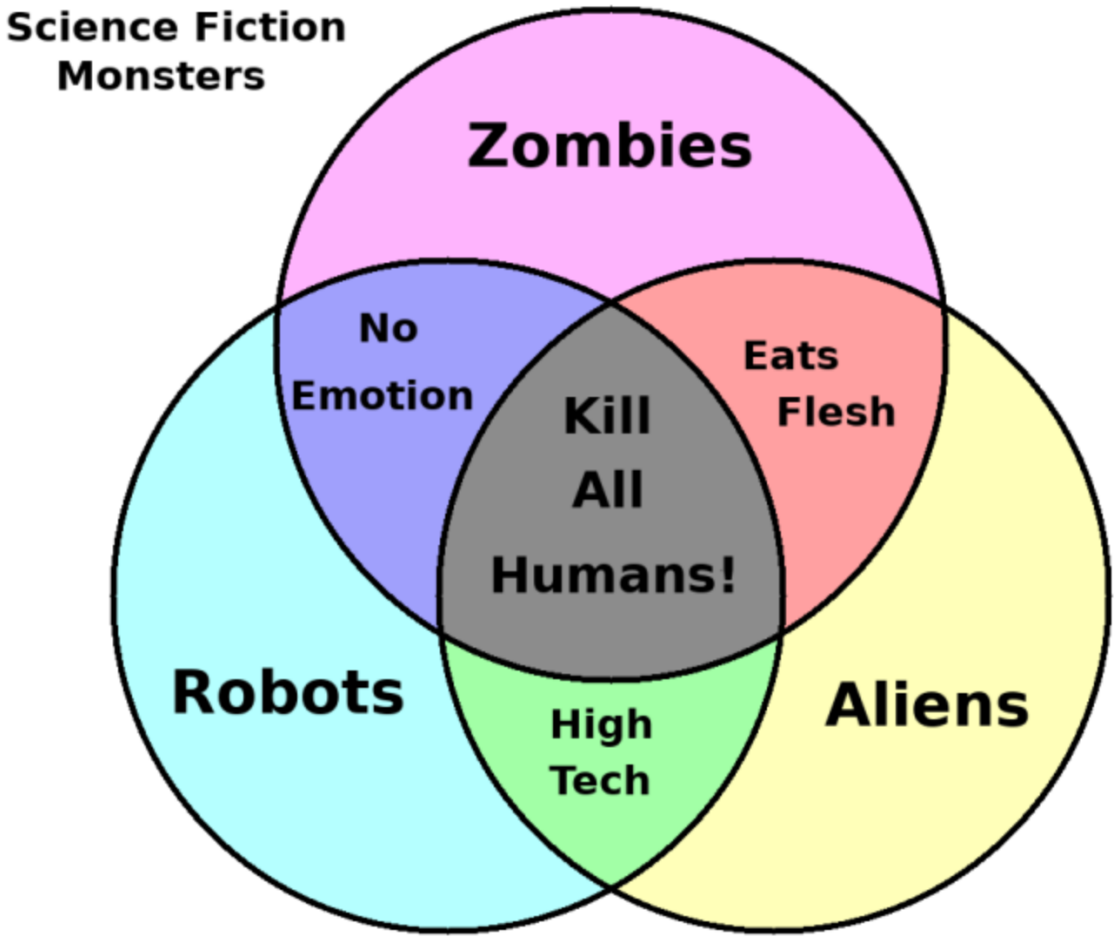
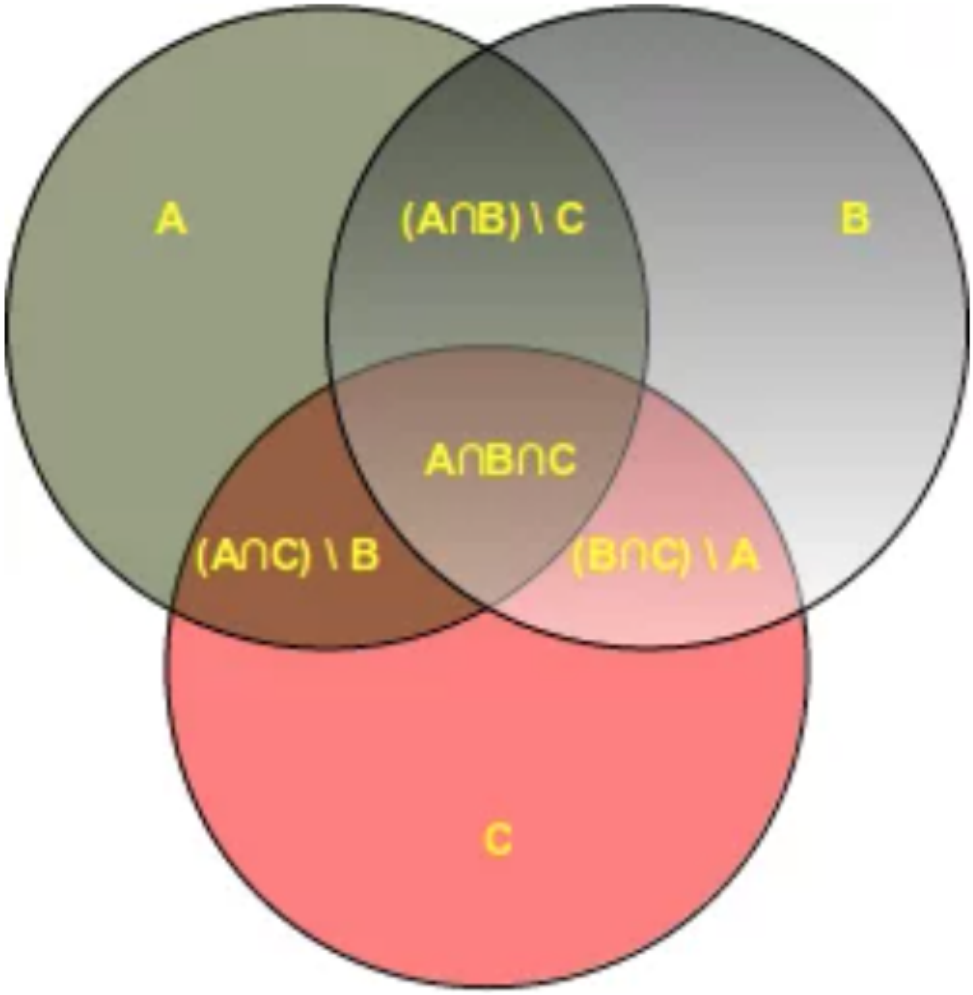


Structured data type: SET


Science Fiction
Monsters
Zombies
No
Emotion
Eats
Flesh
Kill
All
Humans!
Robots
High
Tech
Aliens



Introduction

Set is an un-ordered collection of immutable elements with no duplicate elements in it.

- All the elements in set are `unique`
- Set does **NOT** contain `duplicates`
- Set is an `un-ordered`
- Sets do not support `indexing` and `slicing`

- Set is `mutable` data structure
- Set can only contain `immutable` objects

To create sets that are immutable we have `frozenset`

Use cases

- Membership testing (fast)
- Removing duplicates

Creating set

How to create empty set?

How to create set with values?

```
- You can pass any type of collection
to `set()` constructor.

- When string is passed to `set()`
constructor, it gets singularized into
seperate characters.
```

Codebase

```
>>> foo = set()
>>> type(foo) # `foo` is object of class `se
<class 'set'>
```

```
# Set using curly brackets

>>> foo = {1, 2, True, 100.50, "banana"}
>>> type(foo)
<class 'set'>


# Using tuple
>>> example1 = set((1, 2, 3, "foo"))
{'foo', 1, 2, 3}


# using list
>>> example2 = set([20.50, True, 10])
{True, 10, 20.5}


# using string
example3 = set("prashant")
{'s', 't', 'h', 'n', 'a', 'r', 'p'}

# NOTE :: Try doing `type()` on each object.
```

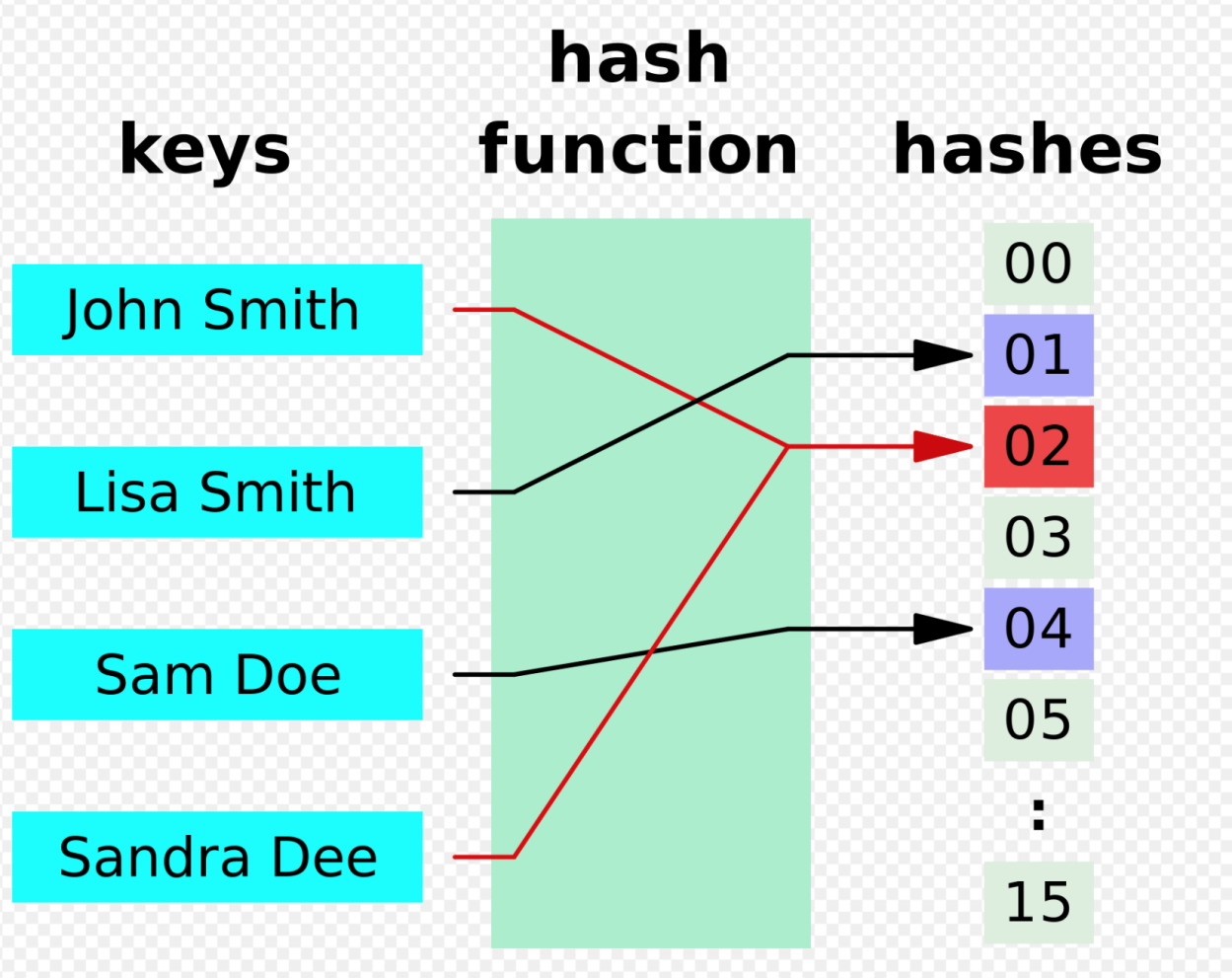
`dir()` on Set

```
>>> dir(set())

['__and__',
...
...
 '__eq__',
 '__hash__',
...
...
...
 'add',
 'clear',
 'copy',
 'difference',
 'difference_update',
 'discard',
 'intersection',
 'intersection_update',
 'isdisjoint',
 'issubset',
 'issuperset',
 'pop',
 'remove',
 'symmetric_difference',
 'symmetric_difference_update',
 'union',
 'update']
```

Hashing

It is s a concept in computer science which is used to create high performance, pseudo random access data structures where large amount of data is to be stored and accessed quickly.



- All the elements in set are `hashable`.
- `Hashable` means the hash value of the object does not change during its life time.
- All immutable data types in Python are `hashable`.
- data types that have `__hash__` and `__eq__` methods defined are `hashables`

Frozenset

`Frozenset` is similar to set but it is immutable.

```
>>> rivers = frozenset(["Narmada", "Sindhu", "Ganga"])
>>> rivers.add("Godavari")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'frozenset' object has no attribute 'add'

>>> # Meaning, fozenset will not have any attributes to modify itself.
```

Try performing `dir()` operation on top of `Frozenset` object and see what all attributes are available.

```
>>> rivers = frozenset(["Narmada", "Sindhu", "Ganga"])
>>> dir(rivers)
['__and__',
 ...
 ...
 ...
 ...
 'copy',
 'difference',
 'intersection',
 'isdisjoint',
 'issubset',
 'issuperset',
 'symmetric_difference',
 'union']
```

Set operations

- `isdisjoint`

Return `True` if the set has no elements in common with *other*.

Sets are disjoint **if and only if** their intersection is an empty set.

```
>>> example1 = {1, 2, 3}
>>> example2 = {3, 4, 5, 6}
>>> example1.isdisjoint(example2)
False
```

- `issubset` [`set <= other`]
  - Test whether every element in the set is in *other*.
  - `set < other` Test whether the set is a proper subset of *other.* That also means, `set <= other and set != other`.

```
>>> states = {"MH", "KA", "Delhi"}
>>> all_states = {"MH", "KA", "Delhi", "Asam", "Gujrat"}
>>> states.issubset(all_states)

>>> states <= all_states
True

>>> states <= all_states and states!=all_states
True
```

- `issuperset` [`set >= other`]
  - Test whether every element in *other* is in the set.
  - `set > other` Test whether the set is a proper superset of *other*, that also means, `set >= other and set != other`.

```
# superset
# > proper superset
# >= normal superset
>>> country = {"india", "pakistan", "usa"}
>>> world = {"india", "pakistan", "usa", "england", "austrelia"}
>>> world.issuperset(country)
True

>>> world > country
True
```

- `union` `set | other | ...`
  - Return a new set with elements from the set and all others.

```
>>> foo = {1, 2, 3, 4, 5}
>>> bar = {4, 5, 6, 7, 8}
>>> set3 = {5, 6, 10, 20, 30}
>>> foo.union(bar, set3)
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 10, 20, 30}

>>> foo | bar | set3  # using `pipe`
{1, 2, 3, 4, 5, 6, 7, 8, 10, 20, 30}
```

- `intersection` `set & other & ...`
  - Return a new set with elements common to the set and all others.

```
>>> foo = {1, 2, 3, 4, 5}
>>> bar = {4, 5, 6, 7, 8}
>>> result = foo.intersection(bar)  # function has a return value
>>> result

result = foo & bar
result
```

- `difference` `set - other - ...`
  - Return a new set with elements in the set that are not in the others.

```
>>> x.difference(y).difference(z)
>>> x - y - z

>>> example1 = {1, 2, 3, 5, 6}
>>> example2 = {2, 6}
>>> example3 = {1}
>>> result = example1.difference(example2, example3) # func has return value
{3, 5}

>>> example1 - example2 - example3
{3, 5}
```

- `symmetric_difference` `set ^ other`
  - Return a new set with elements in either the set or *other* but not both.

```
>>> example1 = {1, 2, 3, 4, 5, 6}
>>> example2 = {5, 6, 7}

>>> result = example1.symmetric_difference(example2) # return value
```

```
>>> result
{1, 2, 3, 4, 7}
```

- `copy`
  - Return a shallow copy of the set.

```
>>> example1 = {1, 2, 3, 4, 5, 6}
>>> example2 = example1.copy()
>>> example2
{1, 2, 3, 4, 5, 6}
```

NOTE

The following table lists operations available for `set` that do **NOT apply to `frozenset` (making it a immutable data structure):**

- `update` `set |= other | ...`
  - Update the set, adding elements from all others.

```
>>> foo = {1, 2, 4}
>>> bar = {2, 4, 5, 6, 7}
>>> foo.update(bar)
>>> foo
{1, 2, 4, 5, 6, 7}
```

- `intersection_update` `set &= other & ...`
  - Update the set, keeping only elements found in it and all others.

```
>>> foo = {1, 2, 3, 4, 5}
>>> bar = {4, 5, 6, 7, 8}
>>> result = foo.intersection_update(bar)
>>> result      # because `intersection_update` does not return anything
None
```

```
>>> foo         # rather it updates object using which function was invoked
{4, 5}
```

- `difference_update` `set -= other | ...`
  - Update the set, removing elements found in others.

```
>>> example1 = {1, 2, 3, 5, 6}
>>> example2 = {2, 6}
>>> example3 = {1}
>>>
>>> result = example1.difference_update(example2, example3)
>>> result  # because `difference_update` does not return anything
None

>>> example1   # rather it updates object using which function was invoked
{3, 5}
```

- `symmetric_difference_update` `set ^= other`
  - Update the set, keeping only elements found in either set, but not in both.

```
>>> example1 = {1, 2, 3, 4, 5, 6}
>>> example2 = {5, 6, 7}
>>> result = example1.symmetric_difference_update(example2) # return value
>>> result    # because `intersection_update` does not return anything
None
>>> example1    # rather it updates object using which function was invoked
{1, 2, 3, 4, 7}
```

- `add`
  - Add an element to the set.

```
# add
>>> foo = set()
>>> foo.add(10)
>>> foo
{10}

>>> foo.add(10) # duplicate addition
```

```
>>> foo
{10}
```

- `remove`
  - Remove an element from the set.
  - Raises `KeyError` if *element* is not contained in the set.
  - Alternatively, we can use `discard()` method

```
>>> example.remove(elem)
# KeyError if element does not exist
# Alternatively, we can use discard() method
```

- `discard`
  - Remove an element from the set if it is present.
  - Return `None` if element does not exist in a given set

```
>>> example1 = {1, 2, 3, 5, 6}
>>> example1.discard(6)
>>> example1
{1, 2, 3, 4, 6}

>>> # What if the element does **NOT** exist in the set.
>>> result = example1.discard(900)
>>> result
None
```

- `pop`
  - Remove and return a random element from the set.
  - Raises `KeyError` if the set is empty.

```
>>> example1 = {1, 2, 3, 4, 5, 6}
>>> example1.pop()
1
```

```
>>> # `pop()` will return random element in each execution
>>> help(example1.pop)

Help on built-in function pop:

pop(...) method of builtins.set instance
    Remove and return an arbitrary set element.
    Raises KeyError if the set is empty.
```

- `clear`
  - Remove all elements from the set.

```
>>> example1 = {1, 2, 3, 4, 5, 6}
>>>
>>> example1.clear()
>>> example1
set()
```