1811/2807/7001ICT
Programming Principles

School of Information and Communication Technology
Griffith University

Trimester 1, 2024

23 Python's Class set

Python's `set` class is a powerful collection type.

23.1 Class set

A Python `set` is a collection of values, where the values are unique (no duplicates).

A `set` is *mutable*, meaning we can add and remove objects to/from a set.

But to be able to ensure that the objects in a `set` are unique, and will always stay unique, they must be *immutable*.

So `set`s may contain, numbers, strings, `bool`s, tuples, but not lists or `set`s (or `dict`s).

23.1.1 Empty sets

The *only* way to write an empty set in Python is to call the constructor.

You might have guessed `{}` would make an empty set, but because dictionaries are used more often, and they also use braces, `{}` makes an empty `dict`.

```
>>> set()
set()
>>> {}
{}
>>> type({})
<class 'dict'>
>>>
```

23.1.2 Non-empty sets

Non-empty sets may be formed:

- with set literals, using braces;

```
>>>  {1, 3, 'Andrew'}
{1, 3, 'Andrew'}
>>>
```

- by adding elements to existing sets;

```
>>> a = {1, 2}
>>> a.add(5)
>>> a
{1, 2, 5}
>>>
```

- using the `set` constructor with another collection or sequence as its argument; and

```
>>> b = set([1, 2, 2, 2, 3])
>>> b
{1, 2, 3}
>>>
```

- with `set` comprehensions.

```
>>> {i for i in range(10) if i % 3 == 0}
{0, 9, 3, 6}
>>>
```

Note the weird order of the elements in the last example. The order of elements in a set is not important or preserved.

23.1.3 Set operations

These are things that can be done with `set`s.

Python	*math*	*description*
`set()`	$\{\}$	Make an empty set.
`set(`*`sequence`*`)`		Make a set from the values in the sequence.
`s.add(x)`		Add `x` to `s`.
`s.remove(x)`		Remove `x` from `s`.
`len(s)`	$\#S$	Count the number of elements in `s`.
`x in s`	$x \in S$	Is `x` an element of `s`?
`x not in s`	$x \notin S$	Is `x` not an element of `s`?

Python	*math*	*description*
`s.union(t)`, or `s | t`	$S \cup T$	Form the union of `s` and `t`.
`s.intersection(t)`, or `s & t`	$S \cap T$	Form the intersection of `s` and `t`.
`s.difference(t)`, or `s - t`	$S - T$	Form the difference of `s` and `t`.
`s.issubset(t)`, or `s <= t`	$S \subseteq T$	Is `s` a subset of `t`?
`s < t`	$S \subset T$	Is `s` a proper subset of `t`?
`s.issuperset(t)`, or `s >= t`	$S \supseteq T$	Is `s` a superset of `t`?
`s > t`	$S \supset T$	Is `s` a proper superset of `t`?

Python	*math*	*description*
`s.isdisjoint(t)`	$S \cap T = \{\}$	Are `s` and `t` disjoint?
`s.pop()`		Return and remove a random element from `s`.
`s.clear()`		Remove all elements from `s`.

There are also some shorthand assignment operators for sets: `|=`; `&=`; and `-=`.

23.2 Set example

This is a classic problem that needs a `set`.

> *Problem:* Write a program that prompts for and reads a file name, then reads the file, and prints all the distinct words in the file, in ascending order.

Hint: use the `sorted` built-in function.

Section summary

This section covered:

- The `set` class, set literals, and operations on sets.