

Sets and Dictionaries

Exercises

Week 7

Prior to attempting these exercises ensure you have read the lecture notes and/or viewed the video, and followed the practical. You may wish to use the Python interpreter in interactive mode to help work out the solutions to some of the questions.

Download and store this document within your own filespace, so the contents can be edited. You will be able to refer to it during the test in Week 6.

Enter your answers directly into the highlighted boxes.

For more information about the module delivery, assessment and feedback please refer to the module within the MyBeckett portal.

Specify two ways in which a Set varies from a List.

Answer:

The two ways in which a Set varies from a list are:

1. Uniqueness of element.
2. Order of element.

Write a Python statement that uses the `set()` *constructor* to produce the same Set as the following -

```
languages = { "C++", "Java", "C#", "PHP", "JavaScript" }
```

Answer:

```
languages = set(["C++", "Java", "C#", "PHP", "JavaScript"])
```

Is a Set **mutable** or **immutable**?

Answer:

A set is mutable.

Why does a Set not support *indexing* and *slicing* type operations?

Answer:

A Set does not support indexing and slicing type operations because the elements in sets do not have fixed position.

Why is a `frozenset()` different from a regular set?

Answer:

A `frozenset()` is different from a regular set because it is immutable, meaning elements cannot be changed, added or removed.

How many elements would exist in the following set?

```
names = set("John", "Eric", "Terry", "Michael", "Graham", "Terry")
```

Answer:

The code will raise a type error cause set() takes single iterable and not multiple arguments.

And how many elements would exist in this set?

```
vowels = set("aeiou")
```

Answer:

There would be 5 elements in the set vowels: {'a', 'e', 'i', 'o', 'u'}.

What is the name given to the following type of expression which can be used to programmatically populate a set?

```
chars = {chr(n) for n in range(32, 128)}
```

Answer:

The name given to that type of expression is a set comprehension.

What **operator** can be used to calculate the intersection (common elements) between two sets?

Answer:

The operator used to calculate the intersection (common elements) between two sets is the & operator.

What **operator** can be used to calculate the difference between two sets?

Answer:

The operator used to calculate the difference between two sets is the - operator.

What would be the result of each of the following expressions?

`{ "x", "y", "z" } < { "z", "u", "t", "y", "w", "x" }`

Answer:

The result of the given expression would be True, because every element in the first set is also present in second set.

`{ "x", "y", "z" } < { "z", "y", "x" }`

Answer:

The result of the expression `{ "x", "y", "z" } < { "z", "y", "x" }` would be False, because the sets are equal, not a proper subset.

`{ "x", "y", "z" } <= { "y", "z", "x" }`

Answer:

The result of the expression `{ "x", "y", "z" } <= { "y", "z", "x" }` would be True, because the sets are equal.

`{ "x" } > { "x" }`

Answer:

The result of the given expression would be False, because the sets are equal, not a proper superset.

`{ "x", "y" } > { "x" }`

Answer:

The result of `{ "x", "y" } > { "x" }` is True because the first set is a proper superset of the second set.

`{ "x", "y" } == { "y", "x" }`

Answer:

The result is True because sets are unordered, and their equality depends on having the same elements, regardless of the order.

Write a Python statement that uses a **method** to perform the equivalent of the following operation -

```
languages = languages | { "Python" }
```

Answer:

update() method can be used to perform the equivalent of the following operations.
languages.update({ "Python" })
This adds python to the language set.

Do the elements which are placed into a set always remain in the same position?

Answer:

No, the elements in a set do not always remain in the same position because sets are unordered collections.

Is the following operation a **mutator** or an **accessor**?

```
languages &= oo_languages
```

Answer:

The following operation is a mutator.

What term is often used to refer to each *pair* of elements stored within a **dictionary**?

Answer:

The term key-value pair is used to refer to each pair of elements stored within a dictionary.

Is it possible for a dictionary to have more than one **key** with the same value?

Answer:

Yes, it is possible for a dictionary to have more than one key with the same value. Each key in a dictionary must be unique, but multiple keys can map to the same value.

Is it possible for a dictionary to have the same **value** appear more than once?

Answer:

Yes, it is possible for a dictionary to have the same value appear more than once. While the keys in a dictionary must be unique, the values can be duplicated across different keys.

Is a Dictionary **mutable** or **immutable**?

Answer:

A dictionary is mutable.

Are the **key** values within a dictionary **mutable** or **immutable**?

Answer:

The key values in the dictionary are immutable.

How many *elements* exist in the following dictionary?

```
stock = {"apple":10, "banana":15, "orange":11}
```

Answer:

The given dictionary contains 3 elements, where each element is a key-value pair.

And, what is the data-type of the **keys**?

Answer:

The data type of the keys is string (str).

And, what output would be displayed by executing the following statement -

```
print(stock["banana"])
```

Answer:

The output will be 15.

Write a Python statement that uses the `dictionary()` *constructor* to produce the same dictionary as the following -

```
lang_gen = { "Java":3, "Assembly":2, "Machine Code":1 }
```

Answer:

```
lang_gen = dict([("Java", 3), ("Assembly", 2), ("Machine Code", 1)])
```

Now write a simple expression that tests whether the word "Assembly" is a member of the dictionary.

Answer:

```
"Assembly" in lang_gen
```

Write some Python code that uses a `for` statement to iterate over a dictionary called `module_stats` and print only its **values** (i.e. do not output any keys) -

Answer:

```
module_stats = {"module1": 85, "module2": 90, "module3": 75}

for value in module_stats.values():
    print(value)
```

Now write another loop which prints the only the **keys** -

Answer:

```
module_stats = {"Math": 90, "Science": 85, "History": 88}

for key in module_stats.keys():
    print(key)
```

Is it possible to construct a dictionary using a **comprehension** style expression, as supported by lists and sets?

Answer:

```
Yes, it is possible to construct a dictionary using a comprehension style expression, known as a dictionary comprehension.
```

When a Dictionary type value is being passed as an argument to a function, what characters can be used as a prefix to force the dictionary to be **unpacked** prior to the call being made?

Answer:

To unpack a dictionary when passing it as an argument to a function, you can use the double asterisk (**) prefix.

Exercises are complete

Save this logbook with your answers. Then ask your tutor to check your responses to each question.