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:

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:

Mutable.

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

Answer:

A **Set** does not support indexing or slicing because it is unordered, meaning elements have no fixed position.

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

Answer:

A `frozenset` is different from a regular set in that it is immutable, meaning its elements cannot be added, removed, or changed after creation.

How many elements would exist in the following set?

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

Answer:

The code will raise a **TypeError** because set() takes a single iterable, not multiple arguments.

And how many elements would exist in this set?

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

Answer:

The set vowels=set(“aeiou”)will contain **5 unique elements**: {‘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 expression is called 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 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:

In this case, the result would be True because all elements of the first set `{ "x", "y", "z" }` are present in the second set `{ "z" , "u", "t", "y", "w", "x" }`

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

Answer:

The result is False because the sets are identical, not a strict subset.

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

Answer:

The result is True because the <= operator checks for a subset, and the sets are identical.

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

Answer:

The result of the expression `{ "x" } > { "x" }` would be `False`, because the sets are equal, not a proper superset.

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

Answer:

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

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

Answer:

The result of the expression `{ "x", "y" } == { "y", "x" }` is True, because sets are unordered collections, and equality is based on having the same elements regardless of order.

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

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

Answer:

languages.update({ "Python" })

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

Answer:

No, elements in a set do not maintain a fixed 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. In a dictionary, the keys must be unique, but multiple keys can point 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 keys within a dictionary are immutable.

How many *elements* exist in the following dictionary?

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

Answer:

The dictionary stock has 3 elements, each representing a key-value pair.

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

Answer:

The data type of the keys in the stock dictionary is **string**

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

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

Answer:

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

 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}  # Example dictionary

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, similar to list and set comprehensions.

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 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.